目　　录

全国青少年信息学奥赛培训教程

第一章　初识 Pascal 语言

一、Pascal 语言概述

PASCAL 语言也是一种算法语言，它是瑞士苏黎世联邦工业大学的 N. 沃思(Niklaus Wirth)教授于 1968 年设计完成的，1971 年正式发表。1975 年，对 PASCAL 语言进行了修改，作为"标准 PASCAL 语言"。

PASCAL 语言是在 ALGOL 60 的基础上发展而成的。它是一种结构化的程序设计语言，可以用来编写应用程序。它又是一种系统程序设计语言，可以用来编写顺序型的系统软件（如编译程序）。它的功能强、编译程序简单，是 70 年代影响最大一种算法语言。

二、Pascal 语言的特点

从使用者的角度来看，PASCAL 语言有以下几个主要的特点：

1.它是结构化的语言。PASCAL 语言提供了直接实现三种基本结构的语句以及定义"过程"和"函数"(子程序)的功能。可以方便地书写出结构化程序。在编写程序时可以完全不使用 GOTO 语句和标号。这就易于保证程序的正确性和易读性。PASCAL 语言强调的是可靠性、易于验证性、概念的清晰性和实现的简化。在结构化这一点上，比其它（如 BASIC,FORTRAN77）更好一些。

2.有丰富的数据类型。PASCAL 提供了整数、实型、字符型、布尔型、枚举型、子界型以及由以上类型数据构成的数组类型、集合类型、记录类型和文件类型。此外，还提供了其它许多语言中所没有的指针类型。沃思有一个著名的公式："算法＋数据结构＝程序"。指出了在程序设计中研究数据的重要性。丰富的数据结构和上述的结构化性质，使得 PASCAL 可以被方便地用来描述复杂的算法，得到质量较高的程序。

3.能适用于数值运算和非数值运算领域。有些语言（如 FORTRAN 66,ALGOL 60）只适用于数值计算，有些语言（如 COBOL ）则适用于商业数据处理和管理领域。PASCAL 的功能较强，能广泛应用于各种领域。PASCAL 语言还可以用于辅助设计，实现计算机绘图功能。

4.PASCAL 程序的书写格式比较自由。不象 FORTRAN 和 COBOL 那样对程序的书写格式有严格的规定。PASCAL 允许一行写多个语句，一个语句可以分写在多行上，这样就可以使 PASCAL 程序写得象诗歌格式一样优美，便于阅读。

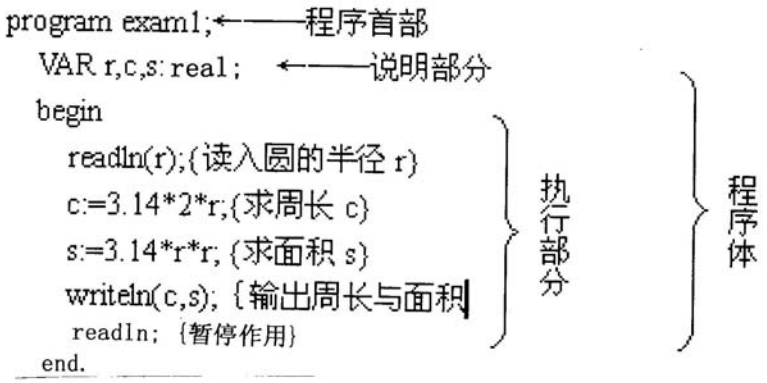
由于以上特点，许多学校选 PASCAL 作为程序设计课程中的一种主要的语言。它能给学生严格而良好的程序设计的基本训练。培养学生结构化程序设计的风格。但它也有一些不足之处，如它的文件处理功能较差等。

三、Pascal 语言程序的基本结构

任何程序设计语言都有着一组自己的记号和规则。PASCAL 语言同样必须采用其本身所规定的记号和规则来编写程序。尽管不同版本的 PASCAL 语言所采用的记号的数量、形式不尽相同，但其基本成分一般都符合标准 PASCAL 的规定，只是某些扩展功能各不相同罢了。下面我们首先来了解 Pascal 语言的程序基本结构。

为了明显起见先举一个最简单的 PASCAL 程序例子：

【例 1 】输入半径 r,求圆的周长和面积。



从这个简单的程序可以看到：

1. 一个 PASCAL 程序分为两个部分：程序首部和程序体（或称分程序）。

2. 程序首部是程序的开头部分，它包括：

⑴程序标志。用"program"来标识"这是一个 PASCAL 程序"。PASCAL 规定任何一个 PASCAL 程序的首部都必须以此字开头。在 turbo pascal 语言中，首部也可省略。

⑵程序名称。由程序设计者自己定义，如例中的 exam1。

在写完程序首部之后，应有一个分号。

3. 程序体是程序的主体，在有的书本里也称"分程序"。程序体包括说明部分（也可省略）和执行部分两个部分。

⑴说明部分用来描述程序中用到的变量、常量、类型、过程与函数等。本程序中第二行是"变量说明"，用来定义变量的名称、类型。

PASCAL 规定，凡程序中用到所有变量、符号常量、数组、标号、过程与函数、记录、文件等数据都必须在说明部分进行定义（或称"说明"）。也就是说，不允许使用未说明先使用。

⑵执行部分的作用是通知计算机执行指定的操作。如果一个程序中不写执行部分，在程序运行时计算机什么工作也不做。因此，执行部分是一个 PASCAL 程序的核心部分。

执行部分以"begin"开始，以"end"结束，其间有若干个语句，语句之间以分号隔开。
执行部分之后有一个句点，表示整个程序结束。

4.PASCAL 程序的书写方法比较灵活。当然，书写不应以节省篇幅为目的，而应以程序结构清晰、易读为目的。在编写程序时尽量模仿本书中例题程序格式。

5.在程序中，一对大括号间的文字称为注释。注释的内容根据需要书写，可以用英语或汉语表示。注释可以放在任何空格可以出现的位置。执行程序时计算机对注释不予理睬。

四、Turbo Pascal 语言系统的使用

目前，常用的 Pascal 语言系统有 Turbo Pascal7.0 与 Borland Pascal 7.0，下面我们就来学习 Turbo Pascal 7.0 系统的使用。

1. 系统的启动

在运行系统目录下的启动程序 TURBO.EXE，即可启动系统。屏幕上出现如图 1 所示的集成环境。

2. Turbo Pascal 系统集成环境简介

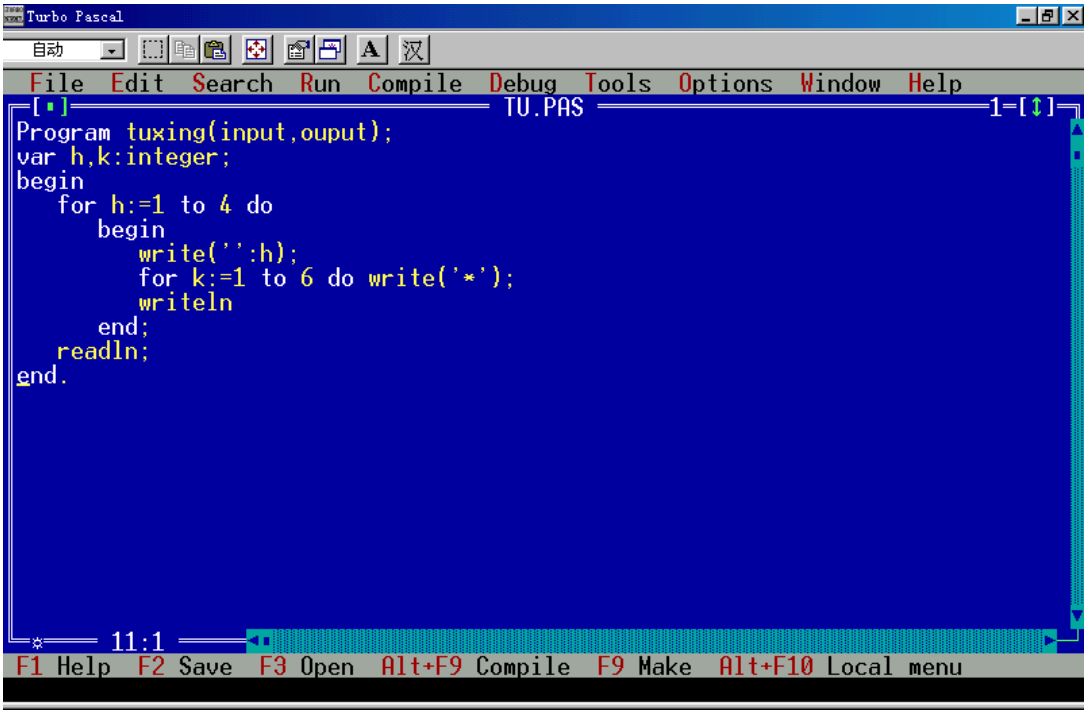
最顶上一行为主菜单。中间蓝色框内为编辑窗口，在它个编辑窗口内可以进行程序的编辑。最底下一行为提示行，显示出系统中常用命令的快捷键，如将当前编辑窗口中文件存盘的命令快捷键为 F2，获得系统帮助的快捷键为 F1，等等。

3．新建程序窗口

按 F10 进行主菜单，选择 FILE 菜单，执行其中 New 命令。就可建立一个新的程序窗口（默认文件名为 Noname00.pas 或 Noname01.pas 等）。

4．程序的输入、编辑

在当前程序窗口中，一行一行的输入程序。事实上，程序窗口是一个全屏幕编辑器。所以对程序的编辑与其它编辑器的编辑方法类似，这里不再重复。



5.编译程序

当程序输入完毕之后，一般要先按 Alt+F9（或执行 compile 菜单中 compile 命令）对程序进行编译。如果程序有语法错误，则会在程序窗口的第一行处显示第一个红色错误信息。若无语法错误，则窗口正中央会出现一个对话框，提示编译成功。接下来，我们可以运行程序了。如果在编译过程中发现程序有语法错误，系统会提示第一个错误信息。如"；'expected"，提示缺少分号；"）'expected"，提示缺少右括号。此时，应有针对性的进行修改。修改后，再重复编译的过程，直到编译成功。

6.运行程序

程序的运行可以通过按 ALT+R 打开 RUN 菜单中的 RUN 命令，或直接按快捷键 CTRL+F9。则可以在用户窗口中输出运行结果。通常在程序运行结束后系统回到 Pascal 系统的集成环境，因此要查看运行结果，要按 ALT+F5 将屏幕切换到用户屏幕。

下面是该程序的运行结果

```
******
  ******
    ******
      ******
```

7．程序的保存与打开

选择主菜单 File 中的菜单项 Save，或按快捷键 F2，在出现的如图的对话框中输入文件名：tu.pas，单击""OK，则程序就以 tu.pas 为文件名保存在当前目录中了。

第二章　简单程序设计

我们学习了 Pascal 语言的程序基本结构，在一个程序中，所有的操作都由执行部分来完成，而执行部分又都是由一个个语句组成的。因此，下面开始我们要学习 pascal 语言的基本语句，并且在学习过程中逐步学会程序设计的基本方法。

这节课我们要学习两种语句，即赋值语句与输出语句。在语句学习之前我们要先了解一些 pascal 语言的基础知识。

第一节　数据类型、常量、变量

一、 常量、变量与算术表达式

（一）常量

在程序运行过程中，其值不能被改变的量称为常量。如 123,145.88,'abc',true 等。

1.整型常量

整型常量采用我们平常使用的十进制整数表示。如 138,0,-512 等都是整型常量，而 18.或 18.0 都不是整型常量。

pascal 中有一个标准标识符 Maxint，它代表所使用的计算机系统允许的最大整型数，而最小的整型数即为－Maxint-1。

Turbo Pascal 还定义了长整数常量 MaxLongInt，其值为 2147483647。

2.实型常量

实型常量包括正实数、负实数和实数零。pascal 中表示实型常量的形式有两种。

⑴十进制表示法

这是人们日常使用的带小数点的表示方法。

如 0.0,-0.0,+5.61,-8.0,-6.050 等都是实型常量，而 0.,.37 都不是合法的实数形式。

⑵科学记数法

科学记数法是采用指数形式的表示方法，如 1.25×10^5 可表示成 1.25E+05。在科学记数法中，字母“E”表示 10 这个“底数”，而 E 之前为一个十进制表示的小数，称为尾数，E 之后必须为一个整数，称为“指数”。

如-1234.56E+26 ， +0.268E-5 ， 1E5 是合法形式，而.34E12 ， 2.E5 ， E5 ，E,1.2E+0.5 都不是合法形式的实数。

无论实数是用十进制表示法还是科学表示法，它们在计算机内的表示形式是一样的，总是用浮点方式存储。

和整数相比，实数能表示的范围大得多，但值得注意的是实数的运算整数的运算速度慢且无法像整数那样精确表示，只能近似表示。

3.字符常量

在 Pascal 语言中，字符常量是由单个字符组成，所有字符来自 ASCII 字符集，共有 256 个字符。在程序中，通常用一对单引号将单个字符括起来表示一个字符常量。如：’a’，’A’，’0’等。特殊地，对于单引号字符，则要表示成’’’’。对于 ASCII 字符集中，按每个字符在字符集中的位置，将每个字符编号为 0－255，编号称为对应字符的序号。

4.布尔常量

布尔型常量仅有两个值，真和假，分别用标准常量名 true 和 false 表示。它们的序号分别为 1 和 0。

5.符号常量

一个常量即可以直接用字面形式表示（称为直接常量， 如 124,156.8），也可以用一个标识符来代表一个常量，称为“符号常量”。但符号常量必须在程序中的说明部分定义，也就是说先定义，后使用。

定义符号常量的一般格式:

```
CONST
  <常量标识符>=<常量>
```

说明：常量说明部分以关键字 const 开头，后面的标识符为常量标识符，其中“=”号后的常量为整数、实数、字符、 字符串（字符、字符串常量在后面章节中将作介绍）。而且，在常量说明部分可以将几个常量说明成符号常量，共用一个关键字“const”。例如:

```
program ex;
const
    pi=3.14159;
    zero=0;
```
常量说明
```
begin
  write('Enter r=');readln(r);
  c:=2*pi*r;
  s:=pi*r*r;
  writeln('c=',c);
  writeln('s=',s);
end.
```

则在本程序中 pi 和 zero 作为符号常量,分别代表实数 3.14159 和整数 0。也就是说，常量说明部分既定义了常量名及其值，又隐含定义了常量的类型。

关于符号常量，应注意下列几点：

⑴符号常量一经定义，在程序的执行部分就只能使用该常量标识符，而不能修改其值。

⑵使用符号常量比直接用数值更能体现“见名知义”的原则，也便于修改参数，故一个较好的程序中，应尽量使用符号常量，在执行部分基本上不出现直接常量。

（二）变量

变量代表了一个存储单元，其中的值是可变的，故称为变量。如游戏“魂斗罗”中玩者命的个数最初为 3，当你死了一次命减少一，这里命的个数就是一个变量（或者说命的个数存储在一个存储单元中）。即在程序运行过程中，其值可以改变的量，称为变量。

变量有三个要素是：变量名、变量类型、变量值。

一个程序中可能要使用到若干个变量，为了区别不同的变量，必须给每个变量（存贮单元）取一个名（称为变量名），该变量（存贮单元）存放的值称为变量的值，变量中能够存放值的类型为变量的类型。例如 “魂斗罗”游戏中用于存放“命”的变量,在游戏程序中的名字可取为 N,它的类型为整型,游戏初始时这个变量的值为 3。

1. 变量名

用一个合法的标识符代表一个变量。如 n,m,rot,total 等都是合法变量名。在程序中用到的变量必须在说明部分加以说明，变量名应遵循自定义标识符的命名规则，并注意“见名知义”的原则，即用一些有意义的单词作为变量名。

“自定义标识符”的命名规则为：自定义标识符必须以字母（包含下划线“_”）开头，后面的字符可以是字母或数字。标识符长度不超过 63 个字符。

2. 变量的类型

常量是有类型的数据，变量在某一固定时刻用来存放一个常量，因此也应有相应的类型。如整型变量用来存放整数，实型变量用来存放实数。

3. 变量说明

在程序中若要使用变量，变量的名称及类型在程序的变量说明部分加以定义，变量的值则在程序的执行部分中才能赋给。

变量说明的一般格式：

```
VAR
    <变量标识符>[,<变量标识符>]:<类型>;
      (中括号内部分表示可省,下同)
```

其中 VAR 是 pascal 保留字,表示开始一个变量说明段, 每个变量标识符或由逗号隔开的多个变量标识,必须在它的冒号后面说明成同一类型。一个程序中，可以说明许多不同类型的变量，每种类型变量之间用分号隔开，共用一个 VAR 符号。

例如:

```
var
  age,day:integer;
  amount,average:real;
```

其中，Integer(整型)、Real(实型)是标准标识符， 它们是“类型标识符”，代表了确定的类型，如 age 和 day 被定义为整型变量，amount 和 average 被定义为实型变量。

一旦定义了变量，就确定了它的类型，也就是说，就确定了该变量的取值范围和对该变量所能进行的运算。

（三）算术表达式

⑴算术表达式的定义

pascal 语言中的算术表达式是由符合 pascal 语法规定的运算对象(包括常量、变量、函数)、算术运算符、圆括号组成的有意义的式子。如：A+3.14159*5/8.4-Abs(-1123)

⑵算术运算符

常用的有以下 6 个算术运算符：

①＋　(加)

②－　(减)

③＊　(乘)

④／　(实数除)得到结果为实型.如 5.0/2.0=2.5，　5/2= 2. 5，4/2=2.0 而不等于 2。

⑤DIV　(整除) DIV 它要求除数和被除数均为整型，结果也为整型。如 10 DIV 2=5,10 DIV 3=3，5 DIV 10=0. -15 DIV 4= -3。DIV 运算只取商的整数部分，参与 DIV 运算的两个对象不能为实型。

⑥mod　(求余)，也只能用于整数运算，结果为整数。例如:10 mod 4=2，-17 mod 4= -1，4 mod (-3)=1，- 4 mod 3= -1，即 a mod b=a-(a div b)*b。

(3)运算优先顺序

如果一个表达式里出现两个或两个以上的运算符，则必须规定它们的运算次序。pascal 规定：

①表达式中相同优先级的运算符,按从左到右顺序计算;

②表达式中不同优先级的运算符,按从高到低顺序计算;

③括号优先级最高,从内到外逐层降低;

在算术运算中运算符的优先顺序与数学上的四则运算一致，即“先乘除后加减”(注：“MOD”、“DIV”运算的优先级与“＊”、“/”相同)。

(四)Pascal 标准函数

例如：

odd(5)为判断自变量是否为奇数，故其值为 true。

Abs(—3)表示绝对值函数，因此其值为 3。

Sqr(5)是求平方函数，故其值为 25。

Sqrt(100)为求平方根函数，故其值为 10。

Chr(48)为求 ASCII 码值，故其值为’0’。

Trunc(1.999)为截尾函数，故其值为 1。

注意：

（1）round(x)是舍入函数，对于正数，舍小数后，函数值比原值要小，入小数后，函数值比原值要大。负数则正好相反。也就是说，正数舍小入大，负数舍大入小。

（2）chr 函数和 ord 函数在字符范围内构成一对反函数，

如：chr(ord(‘a’))=’a’　ord(chr(61))=61

（3）函数和函数构成一对反函数，如：

pred(succ(x))=x　succ(pred(x))=x

（4）x 的 n 次方利用换底公式表示为 exp(n*ln(x))

（5）sin(x)、cos(x)的自变量是弧度，若给出的是角度值，转换公式是：

弧度值=3.1416/180*角度值

（6）ord(true)=1，ord(false)=0

第二节 赋值语句

对程序已经创建的变量，如何取值？通常使用赋值语句来给变量提供数据，它具有计算和赋值的功能，程序中所进行的各种运算，大多数是在赋值语句中实现的。

（1）格式

变量标识符：=表达式

（2）语义

赋值语句的执行是“先计算，后赋值”。即先计算表达式的值，然后将值赋给变量标识符，具有计算和赋值的双重功能。

例如：pi1:=3.1415*6 是计算 3.1415*6 的值，然后将其值赋值给变量 pi1.

[例 1]下面的程序执行后，变量 b、c、d 的值是多少？

```
program p2_1(input,output);
  Const   a=256;
   Var   b,d:integer;
         c:real;
   Begin
      b:=a div 16;{计算表达式 a div16 的值为 16，赋值给变量 b}
      c:=a/b;{计算表达式 a/b 的值，也就是将 a 的值 256 除以 b 的值，结果为 16，
              但是因变量 c 的类型是实型，所以赋予给变量 c 的值应为 16.0}
      d:=a;   {变量 d 的值为 256}
      Readln; {暂停}
   end.
```

（3）说明

1.":="称为赋值号，要注意不能与关系运算符“＝”混淆，只有在赋值语句中才使 用赋值号。赋值号具有方向性，是将赋值号右边表达式的值计算出来，赋予赋值号左边的变量，所以赋值号的左边只能是变量；常量说明中只能用等号，如例 4 －1 。

2.赋值号两边的类型应该相同。只有一点可以例外，那就是当表达式的值为整型时，它可以自动转化成实型后赋给一个实型变量。

3.一个赋值语句只能给一个变量赋值。变量可以进行多次赋值，赋值后的变量将在程序中一直保持不变，直到该变量重新赋值成其他的值。

4.被赋值的变量本身可以作为因子参与运算，如 n:=n-1,i:=i+1,s:=s+x. 为了深入理解赋值语句，请看下面的例子：

[例 2] 写出执行下面的程序后，变量 a、b 的值。

```
program p4_2 (input,output);
 var  a,b:integer;
    begin
       a:=3;
       b:=a;
       b:=a+1;
       a:=a+1;
       b:=b+1;
       Readln; {暂停}
     end.
```

第三节　输出语句（WRITE 语句）

输出语句的作用是将程序运算的结果输出到屏幕或打印机等输出设备。这里通常是指输出到屏幕。

（一）输出语句的两种格式

1、write 语句

格式 Write(表达式 1，表达式 2，……)；

如:write(1,2,3,4);

write(1.2,3.4,5);

write(‘My name is Liping’);

2、writeln 语句

格式:

Writeln(表达式 1，表达式 2，……)或 writeln

(二)输出语句的功能

计算机执行到某一输出语句时，先计算出输出语句中的每个表达式的值，并将每一个表达式的值一个接一个地输出到屏幕上。

Write 语句与 writeln 语句格式上都相似，但它们在功能上有所不同，两个语句的区别在于，write 语句将其后括号中的表达式一个接一个输出后，没有换行。而 writeln 语句则在输出各个表达式的值后换行。

例如以下两个程序段的输出分别为：

write(1,2,3,4);write(5,6);

输出为：

123456

writeln(1,2,3,4);write(5,6);

输出为：

1234

56

（三）、应用例析

[例 3]：某仓库 5 月 1 日有粮食 100 吨，5 月 2 日又调进 20 吨，5 月 3 日卖出库存的 3 分之二，5 月 4 日又调进库存的 3 倍粮食，问该仓库从 5 月 1 日到 5 月 4 日期间每天的粮食分别是多少吨？(输出每天的库存量)

分析：在这个问题中，主要要描述从 5 月 1 日到 5 月 4 日期间仓库的粮食库存量，且易知它是不断变化的。因此我们可以用一个变量 A 来描述仓库的粮食库存量。

程序可写如下：

```
Program  ex1;
  Var  A : integer;
  Begin
    A:=100;Writeln( ‘5/1:’ ,A);
    A:=A+20;Writeln( ‘5/2:’ ,A);
    A:=A div 3; writeln( ‘5/3:’ ,A);
    A:=A *4;  writeln( ‘5/4:’ ,A);
    Readln;
  End.
```

[例 4]：有三个小朋友甲乙丙。甲有 50 粒糖果，乙有 43 粒糖果，丙有 13 粒糖果。现在他们做一个游戏。从甲开始，将自己的糖分三份，自己留一份，其余两份分别给乙与丙，多余的糖果自己吃掉，然后乙与丙也依次这样做。问最后甲、乙、丙三人各有书多少粒糖果？

分析：这个问题中我们关心的是在游戏过程中每个小朋友的糖果个数，且他们所拥有的的糖果数是在变化的。因此可用 a,b,c 三个变量分别存放甲乙丙三个小朋友在某一时刻所拥有的糖果数。对于每人，分糖后，他的糖果数一定为原来的糖果数 div 3（因为分糖过程糖果的数目不一定都刚好分完，用整除恰恰

可以表示多余的糖自己吃掉)。而其他两人则增加与这个小朋友现在拥有的一样的糖果。
程序可写如下：

```
program ex2;
var A,B,C:integer;
  begin
    A:=50;B:=43;C:=13;  {初始时每个小朋友所拥有的糖果数}
    A:=A div 3; B:=B+A;C:=C+A; {甲小朋友分糖果后，每个人拥有的糖果数变化情况}
    B:=B div 3; A:=A+B;C:=C+B; {乙小朋友分糖果后，每个人拥有的糖果数变化情况}
    C:=C div 3; A:=A+C;B:=B+C; {丙小朋友分糖果后，每个人拥有的糖果数变化情况}
    writeln('A=',A,'B=',B,'C=',C); {输出结果}
    readln;
  end.
```

注：上程序中倒数第三行中’A=’表示一个字符串（即用一对单引号括起来的一串字符），对于字符串，输出字符串的内容（即引号内的所得字符，而引号不输出)。

以上程序的运行结果为：

A=51 B=35 C=16

【上机练习 2.3】

1、已知某梯形的上底 A=13，下底 B=18，高 H=9，求它的面积 S。
2、已知某圆的半径 R=139，求该圆的周长 C 与面积 S?
3. 输入长方形的边长 a,b，计算它的面积和周长，输出。

第四节　输入语句

一、写语句的输出格式

在 pascal 语言中输出数据时是可以按照一定格式的，对整数隐含的输出形式为按十进制数形式。对实数的输出，隐含的形式是科学记数法形式（如果不想用科学记数法输出而用小数形式输出，要自己另行定义)。

事实上，输出语句中的每个输出项中的表达式之后可以加上格式说明，若输出项后没有加格式说明，则数据按系统隐含的格式输出，还可加上一定格式符号按特定格式输出。

1.隐含的输出格式

pascal 语言为整型量、实型量、布尔型量和字符串(用一对单引号括起来的字符序列)规定了每种数据所占的宽度(即一个数据占几列），一个数据所占的宽度称为“场宽”或“字段宽”。系统给出的隐含场宽称为标准场宽。每一种 pascal 版本给定的标准场宽不尽相同。下表给出标准 pascal 和 pc 机上两种 pascal 版所规定的标准场宽。

数据类型	标准场宽 标准 pascal	Turbo pascal
integer	10	实际长度
real	22	17
布尔型	10	4 或 5
字符串	串长	串长

在 Turbo Pascal 系统中，对于整型字符串的输出都是按数据本身长度输出，对于布尔型数据（只有 True 和 False 两种值)，TRUE 为 4 列，FALSE 为 5 列，一律采用大写输出。而 real 型数据的输出时，则按 17 列输出，其中第一列为符号位，正号不显示，后四位为“E±nn”， 中间的 12 列为尾数部分。如：

```
writeln(sqrt(75));
```

则输出□8.6602540379E+00。

而 writeln(sqrt(81));

则输出□9.0000000000E+00。

有时，在程序中往往根据实际情况，需要自己定义场宽。

2.指定场宽

在写语句中输出项含有格式符号时,就是为了指定场宽。

⑴指定单场宽.

格式：write(表达式:N)或 writeln(表达式:N),其中 N 为自然数，指定单场宽后，所有数据不再按标准场宽输出，而按指定场宽输出。若数据实际长度小于指定场宽时，则一律“向右靠齐，左留空格”。

如 write(1234:8);write('abcdef':12);

输出结果:

□□□□1234□□□□□□abcdef

对于标准实型数据指定单场宽时，如果场宽大于标准场宽时，右靠齐按标准场宽格式输出 17 位，左留空格。若场宽小于标准场宽时，第一位仍为符号位，最后四位仍为“E±nn”，中间部分为尾数显示部分。如果指定的宽度小于 8 位，则数据按 8 位格式“*.*E±nn ”输出。

⑵指定双场宽

如果输出项是实数时，如果希望输出的实数不用科学记数法输出,而用小数形式输出,可以用指定双场宽方法输出。

双场宽输出格式为：write(实型表达式：m：n)，其中 m 和 n 都是自然数，m 用以指定整个数据所占的宽度,n 指定输出实数的小数位数。

如 ： write(sqrt(75):9:4);

输出:□□□8.6602

如果双场宽不能满足输出数据的最低要求，系统自动突破指定的场宽限制，按实际长度输出。如:write(sqrt(75):5:4)；要使小数点后有 4 位数字，而总场宽为 5，是不可能的(因为还有一个小数点，小数点前面还有一个数字)。它最低限度要有 6 列，即输出为:

8.6602

例 5 写出下列程序在 turbo pascal 下的输出结果.

```
program ex;
const s='abcdefg';
 var  i:integer; r:real;
      c:char;    b:boolean;
 begin
      i:=1234;r:=1234.5678;
      c:='#';b:=true;
      writeln(i,i:6,i:3);
      writeln(r,r:12:5,r:8:5);
      writeln(c,c:5);
      writeln(s,s:10,s:5);
      writeln(b,b:5,b:3);
 end.
```

运行结果如下:

1234□□12341234

□1.2345678000E+03□□1234.567801234.56780

#□□□□#

abcdefg□□□abcdefgabcdefg

TRUE□TRUETRUE

3. 应用例析

例 6：已知 A=253，B=43，输出 A*B 的运算式子。即输出如下：

```
253*43=10879
     253
   *  43
  -----
     759
  +1012
  -----
   10879
```

分析：对于该问题，我们只要控制好输出时右靠齐即可。即前四行的总宽度一样（例如为 12），第五行总宽度比前面少 1。第六、七行总宽度与前四行一样。

参与程序如下：

```
var a,b:integer;
begin
  a:=253;b:=43;
  writeln(a:10,'*',b,'=',a*b);
  writeln(a:12);
  write('*':8);writeln(b:4);
  writeln('--------':12);
  writeln(a*3:12);
  write('+':6);writeln(a*4:5);
  writeln('--------':12);
  writeln(a*b:12);
end.
```

二、 输入语句（读语句）

在程序中变量获得一个确定的值，固然可以用赋值语句，但是如果需要赋值的变量较多，或变量的值经常变化，则使用本节介绍的输入语句——读语句，将更为方便。读语句是在程序运行时由用户给变量提供数据的一种很灵活的输入动作，它有两种格式：

1.读语句的一般格式：

read(<变量名表>);

readln[(<变量名表>)];

其中变量名表是用逗号隔开的若干个变量名组成的。

功能：从标准输入文件(即 INPUT,一般对应着键盘)中读入数据,并依次赋给相应的变量。

说明：①read 和 readln 是标准过程名,它们是标准标识符。

②执行到 read 或 readln 语句时，系统处于等待状态，等待用户从键盘上输入数据，系统根据变量的数据类型的语法要求判断输入的字符是否合法。如执行 read(a)语句，a 是整型变量，则输入的字符为数字字符时是合法的，当输入结束时，则自动将刚接受的一串数字字符转换为整数赋给变量 a。

③在输入数值型（整型或实型）数据时，数据间要用空格或回车分隔开各个数据，输入足够个数的数据，否则仍要继续等待输入，但最后一定要有回车，表示该输入行结束，直到数据足够，该读语句执行结束，程序继续运行。

例 7. 设 a、b、c 为整型变量，需将它们的值分别赋以 10,20,30,写出对应下列语句的所有可能输入格式。

```
Read(a,b,c);
```

解 根据③,即可列出所有可能输入格式

(a)10□20□30←┘

(b)10□20←┘

30←┘

(c)10←┘

20□30←┘

(d)10←┘

20←┘

30←┘

其中“←┘”表示回车键。下同。

④read 语句与 readln 语句的第一个区别是：

read 语句是一个接一个地读数据，在执行完本 Read 语句(读完本语句中变量所需的数据)后,下一个读语句接着从该数据输入行中继续读数据，也就是说，不换行。如：

```
Read(a,b);
Read(c,d);
Read(e);
```

如果输入数据行如下:

1□2□3□4□5□6□←┘

则 a,b,c,d,e 的值分别为 1,2,3,4,5,如果后面无读语句则数据 6 是多余的，这是允许的。

Readln 则不同，在读完本 Readln 语句中变量所需的数据后，该数据行中剩余的数据多余无用，或者说，在读完本 Readln 语句中变量所需数据后，一定要读到一个回车，否则多余的数据无用。

例 8 设要达到例 1 同样的目的,但语句改为:

```
readln(a,b);readln(c)
```

则例 3 中的 4 种输入格式只有(b)(d)是有效的.

⑤readln 语句与 read 语句的第二个区别是:read 后一定要有参数表,而 readln 可以不带参数表,即可以没有任何输入项，只是等待读入一个换行符(回车)。经常用于暂停程序的运行，直到输入一个回车。

例 9 设有下列语句:

```
read(a,b,c);
readln(d,e);
readln;
readln(f,g);
```

其中,所有变量均为整型。再设输入的数据如下：

1□2←┘
3□4□5□6□7□8←┘
9□10←┘
11←┘
12□13←┘

列表给出每个变量的值.

分析：可以假想有一“数据位置指针”，每读一个数据后，指针后移到该数据之后，每执行一个 readln 语句后，指针移到下一个数据行的开头。各变量的值如下表所示。

变量名	a	b	c	d	e	f	g
值	1	2	3	4	5	11	12

⑥为了避免可能出现的错误，建议在程序中按下列原则使用读语句：(A)如果没有特殊需要,在一个程序中尽量避免混合使用 read 语句和 readln 语句;(B)尽量用 readln 语句来输入数据，一个数据行对应一个 readln 语句;(C)由于执行 read 或 readln 语句时，系统不会提供任何提示信息,因此,编程时最好在 readln 语句之前加以适当提示,例如:

```
write('Input a,b,c:');
readln(a,b,c);
```

在执行时,屏幕上显示:

```
Input a,b,c:■
```

其中,“■”为光标。执行 readln 语句后,系统处于待待输入状态，只有输入了所需数据后才继续往下执行。

【上机练习 2.4】

1. 求长方体的面积，长、宽、高的值由键盘输入。
2. 读入摄氏温度 c，写程序将它转换成华氏温度 f 输出。已知 f=9c/5+32
3. 输入三个字符，然后按输入字符次序输出这三个字符，并输出每个字符的序号，最后按与输入字符相反的次序输出这三个字符。(求序号用 ORD 函数)
4. 输入一个三位自然数，把这个数的百位与个位数对调，输出对调后的自然数。
5. 键盘输入两个小数，经过取整操作后，将其整数部分交换值后输出。

第五节　顺序结构程序设计

到目前为止，我们可以用读、写语句和赋值语句编写一些简单的程序。通过阅读这些程序，可以逐步熟悉 pascal 程序的编写方法和应遵循的规则，为以后各章的学习打基础。

例 10、试编一程序,输入一梯形的上底、下底、高，求该梯形的面积。

分析：整个程序分为三段:输入、计算、输出。程序中用 a,b,h 三个变量分别存放梯形的上、下底与高，S 存放面积。 要而使用这些变量都要先说明，程序的执行部分中先输入上、下底与高，接着求面积 S,最后输出结果 S。

源程序如下:

```
program Tixing;                    {程序首部}
  var a,b,h,s:real;                {程序说明部分}
  begin
    write('Input a,b,h:');
    readln(a,b,h);                 {程序执行部分}
    s:=(a+b)*h/2;
    write('s=',s:10:3);
   end.
```

例 11、某幼儿园里，有 5 个小朋友编号为 1，2，3，4，5，他们按自己的编号顺序围坐在一张圆桌旁。他们身上都有若干个糖果，现在他们做一个分糖果游戏。从 1 号小朋友开始，将他的糖果均分三份（如果有多余的，则他将多余的糖果吃掉），自己留一份，其余两份分给他的相邻的两个小朋友。接着 2 号、3 号、4 号、5 号小朋友也这如果做。问一轮后，每个小朋友手上分别有多少糖果。

分析：这道问题与第三节中的例 4 基本一样，只不过这里有 5 位小朋友，且他们初始时糖果的数目不确定。这里用 a,b,c,d,e 分别存放 5 个小朋友的糖果。初始时它们的值改为由键盘输入。其它都与第三节中的例 4 类似。

参考程序如下:

```
program fentang;
  var a,b,c,d,e:integer;
  begin
    write( 'Please Enter init numbers  ');readln(a,b,c,d,e);
    a:=a div 3;b:=b+a;e:=e+a;{1 号均分后，1、2、5 号的糖果数变化情况}
    b:=b div 3;c:=c+b;a:=a+b;{2 号均分后，1、2、3 号的糖果数变化情况}
    c:=c div 3;b:=b+c;d:=d+c;{3 号均分后，2、3、4 号的糖果数变化情况}
    d:=d div 3;c:=c+d;e:=e+d;{4 号均分后，3、4、5 号的糖果数变化情况}
    e:=e div 3;d:=d+e;a:=a+e;{5 号均分后，4、5、1 号的糖果数变化情况}
    {输出结果}
     writeln('a=',a);
     writeln('b=',b);
     writeln('c=',c);
     writeln('d=',d);
     writeln('e=',e);
     readln; {暂停}
   end.
```

例 12、编一程序求半径为 R 的圆的周长与面积?

分析：程序要先输入半径 R,然后求周长 c 和面积 s,最后输出 c 和 s.

源程序如下:

```
program circle;
  const PI=3.14159;
  var r,c,s:real;
  begin
    write('Enter R=');readln(r);
    c:=2*pi*r;
    s:=pi*sqr(r);
    writeln('c=',c:10:2);
    writeln('s=',s:10:2);
  end.
```

在程序中，为了输出实型周长 C 和面积 S 时,按照小数形式输出,采用了指定双场宽格式。

【上机练习 2.5】

1、从键盘输入 a、b、c 求一元二次方程 $ax^2+bx+c=0$ 的两个实数根(不考虑无解的情况)。

2、输出两个自然数相除的商和余数。

分析：设被除数、除数、商和余数，分别为 A，B，C，D，均为变量，且都是整数类型。题中未给出具体的自然数 A、B，可采用键盘输入方式。

① 给出提示，从键盘输入 a，b；

② 显示两数相除的数学形式；

③ 求出 a 除以 b 的商 c；

④ 求出 a 除以 b 的余数 d；

⑤ 紧接等式后面输出显示商和余数。

3、加法计算器：编程由键盘输入两个整数 a 和 b，计算出它们的和并输出到屏幕上。

4、计算某次考试语文、数学、英语和计算机等四科的总成绩与平均成绩。

(请用输入语句从键盘输入各科成绩)

5、交换两个变量的值：由键盘输入两个正整数 A 和 B，编程交换这两个变量的值。

6、有鸡兔同笼，头 30，脚 90，究竟笼中的鸡和兔各有多少只?

分析：设鸡为 J 只，兔为 T 只，头为 H，脚为 F，则:

J+T=30 ①

2*J+4*T=90 ②

解此题暂不必采用数学上直接解方程的办法，可采用“假设条件与逻辑推理”的办法：

假设笼中 30 个头全都是兔，那么都按每头 4 只脚计算，总脚数为(4*H)，与实际脚数（F）之差为(4*H—F)，如果这个差=0，则笼中全是兔（即鸡为 0 只)；如果这个差值 >0，说明多计算了脚数，凡是鸡都多给算了两只脚，用它除以 2 就能得到鸡的只数，处理步骤为:

① J=(4*H—F)/2 {先用脚数差值除以 2 算出鸡的只数}

② T=H—J {再用总头数减鸡数算出兔的只数}

7、五位好朋友相聚。第一位朋友带来了很多糖块赠送给各位朋友，使每人的糖块在各自原有的基础上翻了一倍；接着第二位好友也同样向每人赠送糖块，他同样使每人的糖块在各人已有的数量上翻了一倍；第三、第四、第五位好友都照此办理。经过这样的赠送之后，每人的糖块恰好都为 32 块。问各位好友原先的糖块数分别是多少?

第三章　选择结构的程序设计

第一节 如果语句（IF 语句）

在现实生活中，我们每天都要进行根据实际情况进行选择。例如，原打算明天去公园，但如果明天天气不好，将留在家里看电视。所以人也会根据条件进行行为的选择。计算机也会根据不同情况作出各种逻辑判断，进行一定的选择。在这课与下一课中，我们将会发现，我们是通过选择结构语句来实现程序的逻辑判断功能。

一、PASCAL 中的布尔（逻辑）类型

在前面，我们学习了整型(integer)与实型（real）。其中 integer 型数据取值范围为-32768 到 32767 之间所有整数。而 real 型数据取值范围为其绝对值在 10^{-38} 到 10^{38} 之间的所有实数。它们都是数值型的（即值都为数）。布尔型（Boolean）是一种数据的类型，这种类型只有两种值，即“真”与“假”。

1、布尔常量

在 Pascal 语言中“真”用 ture 表示，“假”用 False 表示。所以布尔类型只有 TRUE 与 FALSE 两个常量。

2、布尔变量（BOOLEAN）

如果我们将某些变量说明成布尔型，那么这些变量就是布尔变量，它们只能用于存放布尔值（ture 或 false）。

例如，VAR A，B：BOOLEAN;

3、布尔类型是顺序类型

由于这种类型只有两个常量，Pascal 语言中规定 ture 的序号为 1，false 的序号为 0。若某种类型的常量是有限的，那么这种类型的常量通常都有一个序号，我们称这种类型为顺序类型。如前面我们学过的整型（integer），以及后面要学到的字符型（char）都是顺序类型。

4、布尔类型的输入与输出

a）输出

```
VAR A，B：BOOLEAN;
BEGIN
   A：=TRUE; B：=FALSE;
   WRITELN（A，B）;
END.
TRUEFALSE
```

b）布尔类型变量不能直接用读语句输入

布尔类型变量不能通过读语句给它们提供值。事实上，我们可以通过间接方式对布尔变量进行值的输入。

例如，以下程序是错误的：

```
var a,b,c:Boolean;
begin
  readln(a,b,c);  {错误语句}
  writeln(a,b,c);
end.
```

二、关系表达式与布尔表达式

1、什么是关系表达式

用小括号、>、<、>=、<=、=、<>将两个算术表达式连接起来的式子就称为关系表达式（比较式）。 如：3+7>8,x+y<10,2*7<=13 等都是关系表达式。

2、关系表达式的值

很显然，这几个关系表达式中第一个是正确的，第三个是错误的，而第二个表达式可能是对的，也可能是错的。所以我们很容易发现，这些表达式的值是“对”的或“不对”的（或者说，是“真”的或“假”的），即关系表达式的值为布尔值。表示该比较式两端式子的大小关系是否成立。如 3+2>6 是错的，故它的值为 FALSE。同样，45>=32 是对的,故该表达式的值为 true。

关系表达式用于表示一个命题。如：“m 为偶数”可表示为：m mod 2=0。“n 为正数”可表示为：n>0。

3.布尔运算及布尔表达式

为了表示更复杂的命题，Pascal 还引入三种逻辑运算符：not、and、or。它们分别相当于数学上的“非”、“且”和“或”的意义。

这三个运算符的运算对象为布尔量，其中 not 为单目运算，只有一个运算对象，and 与 or 为双目运算，有两个运算对象。它们的运算真值表如下：

A	b	Not a	a and b	a or b	a xor b
false	false	true	false	false	false
false	true	true	false	ture	true
true	false	false	false	true	true
true	true	false	true	true	false

于是，对于一个关系表达式，或多个关系表达式用布尔运算符连接起来的式子就称为布尔表达式。布尔表达式的值也为布尔值。

如果一个表达式里出现两个或两个以上的运算符，则必须规定它们的运算次序。pascal 规定：

①表达式中相同优先级的运算符,按从左到右顺序计算;

②表达式中不同优先级的运算符,按从高到低顺序计算;

③括号优先级最高,从内到外逐层降低;

对于一个复杂的表达式可能同时包含算术运算、关系运算和逻辑运算以及函数运算。运算的优先顺序为：括号→ 函数 →not→*、/、div、mod、and→+、－、or、xor→ 关系运算。

对于复杂的命题，我们可以用布尔表达式来表示。例如，命题：“m,n 都是偶数或都是奇数”可表示为“(m mod 2=0)and(n mod 2=0) or (m mod 2=1)and(n mod 2=1)”。

三、简单的 IF 语句

1、格式

Ⅰ、IF ＜布尔表达式＞THEN 语句;

Ⅱ、IF ＜布尔表达式＞THEN 语句 1 ELSE 语句 2;

（注意Ⅱ型 IF 语句中语句 1 后无“;”号）

2、功能

Ⅰ、执行 IF 语句时，先计算＜布尔表达式＞的值，若为 TRUE 则执行语句，否则不执行任何操作。

Ⅱ、执行 IF 语句时，先计算＜布尔表达式＞的值，若为 TRUE 则执行语句 1，否则执行语句 2;

3、示例

例 1. 输入一个整数 A，判断是否为偶数。（是输出“YES”否则输出“NO”）。

```
Program ex4_2;
  Var a:integer;
  Begin
    Write( ‘a=’ );readln(a);
    If (a mod 2 =0)then writeln( ‘yes’ )
    Else writeln( ‘no’ );
    Readln;
  End.
```

例 2. 联单华榕超市里卖电池,每个电池 8 角钱，若数量超过 1 0 个，则可打 7.5 折。

```
Program ex4_3;
  Var Num:integer;Price,Total:real;
  Begin
    Write( ‘Num=’ );readln(Num);
    Price:=0.8;
    If Num>10 then Price:=Price*0.75;
    Total:=Num*Price;
    Writeln( ‘Total=’ ,Total:0:2);
    Readln;
  End.
```

例 3. 编写一与电脑猜“红”或“黑”的游戏。

分析：用 1 代表红， 0 代表黑。先由计算机先出答案，然后再由人猜，猜对输出“YOU WIN”否则输出“YOU LOST”。为了模拟猜“红”或“黑”的随意性，程序中需要用到随机函数 random(n)。

函数是什么呢，例如大家都知道|-2|=2，|58|=58，那么|x|=?。

如果我们用 y 表示|x|，那么 $y=|x|=\begin{cases} x, & x\geq 0 \\ -x, & x<0 \end{cases}$ 这里 y=|x|就是一个函数，也就是说函数是一个关于一个或多个自变量（未知量，如上例中的 x）的运算结果。

在 pascal 语言中，系统提供了许多内部函数，其中包括|x|函数，当然它用 abs(x)表示。我们如果要求 x^2-y 的绝对值，可以调用内部函数 abs(x*x-y)即可求得。Random(n)也是一个内部函数，调用它能得到 0~n-1 之间的整数（但它不确定的，或说是随机的）。同时由于函数是一个运算结果，所以函数的调用只能出现在表达式中。

```
Program ex4_3;
Var  Computer,People:integer;
  Begin
    Randomize;
    Computer:=random(2);
    Write( 'You guess (0—Red 1-Black) :' );readln(People);
    If People=Computer then writeln( 'YOU WIN' )
    Else writeln( 'YOU LOST' );
    Readln;
  End.
```

【上机练习 3.1】

1.假设邮局规定寄邮件时若每件重量在 1 公斤以内(含 1 公斤)，按每公斤 1.5 元计算邮费，如果超过 1 公斤时，其超出部分每公斤加收 0.8 元。请编程序计算邮件收费。

2.输入三个正整数，若能用这三个数作为边长组成三角形，就计算并输出该三角形的面积，否则输出 Can't。(组成三角形的条件为：任意两边之和大于第三边)

3.输入一个三位数的整数，将数字位置重新排列，组成一个尽可大的三位数。例如：输入 213，重新排列可得到尽可能大的三位数是 321。

4、输入一个整数，打印出它是奇数还是偶数。

5、某服装公司为了推销产品，采取这样的批发销售方案:凡订购超过 100 套的，每套定价为 50 元，否则每套价格为 80 元。编程由键盘输入订购套数，输出应付款的金额数。

6、从键盘读入一个数，判断它的正负。是正数，则输出“+”，是负数，则输出“-”。

7、判断两个数 a，b，输出较大数的平方值。

8、某市的士费起步价 8 元，可以行使 3 公里。3 公里以后，按每公里 1.6 元计算，输入的士的公里数，请你计算顾客需付费多少元?

第二节　IF 语句的嵌套

四、IF 语句的嵌套

在 if 语句中，如果 then 子句或 else 子句仍是一个 if 语句， 则称为 if 语句的嵌套。

例 4. 计算下列函数

$$y=\begin{cases} 1, & x>0 \\ 0, & x=0 \\ -1, & x<0 \end{cases}$$

分析：根据输入的 x 值，先分成 x>0 与 x≤0 两种情况，然后对于情况 x≤0，再区分 x 是小于 0，还是等于 0。

源程序如下:

```
program ex;
var x:real;
    y:integer;
begin
     wrtie('Input x:');readln(x);
     if x>0  then y:=1    {x>0 时，y 的值为 1}
             else         {x≤0 时}
             if x=0  then y:=0
                     else y:=-1;
      writeln('x=',x:6:2,'y=',y);
end.
```

显然，以上的程序中，在 then 子句中嵌套了一个Ⅱ型 if 语句。当然程序也可以写成如下形式：

```
program ex;
var
   x:real;  y:integer;
begin
  wrtie('Input x:');readln(x);
  if x>=0  then
       if x>0  then y:=1
               else y:=0
  else y=-1;
  writeln('x=',x:6:2,'y=',y);
end.
```

但是对于本题，下面的程序是不对的。

```
y:=0;
if x>=0  then
     if x>0 then y:=1
else y:=-1;
```

明显，从此人的程序书写格式可以看出，他想让 else 与第一个 if 配对，而事实上，这是错的。因为 pascal 规定：else 与它上面的距它最近的 then 配对,因此以上程序段的逻辑意义就与题义不符。

要使上程序段中 else 与第一个 then 配对,应将程序段修改为：

```
y:=0;
if x>=0
  then if x>0
       then y:=1
       else
  else y:=-1;
```

或者

```
y:=0;
if x>=0
    then
      begin
        if x>0 then Y:=1;
       end
    else Y:=-1;
```

【上机练习 3.2】

1.输入某学生成绩，根据成绩的好坏输出相应评语。如果成绩在 90 分以上，输出评语：优秀（outstanding）。如果成绩在 60 分到 90 分之间，输出评语：良好（satisfactory）。如果成绩不足 60 分，输出评语：不及格（unsatisfactory）。

2.输入三角形的三边，判断它是否是直角三角形。

3.给一个不多于三位的正整数，求出它是几位数，并分别打印出各位上的数字。

4.对一批货物征收税金。价格在 1 万元以上的货物征税 5%，在 5000 元以上，1 万元以下的货物征税 3%，在 1000 元以上，5000 元以下的货物征税 2%，1000 元以下的货物免税。编写一程序，读入货物价格，计算并输出税金。

5. 输入三角形的三个边，判断它是何类型的三角形（等边 DB？等腰 DY？一般 YB？）。

6. 输入三个数，按由大到小顺序打印出来。

7. 将字母 A、B、C 、D 或 a、b、c、d 转换成 1、2、3、4，其余的字符转换成 5。

8. 输入三个数 a，b，c，打印出最大者.

第三节 情况语句（CASE 语句）

上面我们知道可以用嵌套的 if 语句实现多分支的选择结构。但是如果分支越来越多时，用嵌套的 if 语句实现多分支就显得繁杂。当多分支选择的各个条件由同一个表达式的不同结果值决定时，可以用 case 语句实现。它的选择过程，很象一个多路开关，即由 case 语句的选择表达式的值，决定切换至哪一语句去工作。因此在分支结构程序设计中，它是一种强有力的手段。在实现多路径分支控制时，用 case 对某些问题的处理和设计，比用 if 语句写程序具有更简洁、清晰之感。

(一)、情况语句的一般形式：

```
case 〈表达式〉 of
   〈情况标号表 1〉：语句 1;
   〈情况标号表 2〉：语句 2;
   :
   〈情况标号表 n〉：语句 n
 end;
```

其中 case、of、end 是 Pascal 的保留字， 表达式的值必须是顺序类型，它可以是整型、布尔型及以后学习的字符型、枚举型和子界型。情况标号表是一串用逗号隔开的与表达式类型一致的常量序列。语句可以是任何语句，包括复合语句和空语句。

(二)、case 语句的执行过程

先计算表达式（称为情况表达式）的值，如果它的值等于某一个常量（称为情况常量，也称情况标号），则执行该情况常量后面的语句，在执行完语句后，跳到 case 语句的末尾 end 处。

(三)、说明

①情况表达式必须是顺序类型的；

②情况常量是情况表达式可能具有的值，因而应与情况表达式具有相同的类型；

③情况常量出现的次序可以是任意的；

④同一情况常量不能在同一个 case 语句中出现两次或两次以上；

⑤每个分语句前可以有一个或若干个用逗号隔开的情况常量；

⑥如果情况表达式的值不落在情况常量的范围内，则认为本 case 语句无效，执行 case 语句的下一个语句。turbo pascal 中增加了一个“否则”的情况,即增加一个 else 子句，但也是可省的。

⑦每个常量后面只能是一个语句或一个复合语句。

例 5 根据 x 的值，求函数 Y 的值：

$$y=\begin{cases} x+1 & ,0<x<100 \\ x-1 & ,100\le x<200 \\ -1 & ,\text{其余} \end{cases}$$

分析：利用 case 语句进行程序设计，关键在于巧妙地构造情况表达式。本例中三种情况可用一个表达式区分出来：Trunc(x/100)。因为 x 在(0～100)之间时表达式值为 0；x 在[100,200)时表达式值为 1 ；其余部分可用 else 子句表示。源程序如下:

```
program ex;
var x,y:real;
begin
  write('Input x:');readln(x);
  case trunc(x/100) of
    0:y:=x+1;
    1:y:=x-1;
    else y:=0;
  end;{end of case}
  writeln('x=',x:8:2,'y=',y:8:2);
end.
```

第四节　综合应用

例 6 输入一个年号,判断它是否是闰年。

分析：判断闰年的算法是：如果此年号能被 400 除尽， 或者它能被 4 整除而不能被 100 整除，则它是闰年。否则，它是平年。源程序如下:

```
var year:integer;
begin
  write('Input year:');readln(year);
  write(year:6);
  if (year mod 400=0 )  then
    writeln('is a leap year.')
  else
    if (year mod 4=0)and(year mod 100<>0)
    then writeln('is a leap year.')
    else writeln('is not a leap year.');
end.
```

例 7、期末，班长小 Q 决定将剩余班费 X 元钱，用于购买若干支钢笔奖励给一些学习好、表现好的同学。已知商店里有三种钢笔，它们的单价为 6 元、5 元和 4 元。小 Q 想买尽量多的笔（多鼓励同学），同时他又不想有剩余钱。请您编程序，帮小 Q 制订出一种买笔的方案。

分析：对于以上的实际问题，要买尽量多的笔，易知都买 4 元的笔肯定可以买最多支笔。因此最多可买的笔为 x div 4 支。由于小 q 要把钱用完，故我们可以按以下方法将钱用完：

若买完 x div 4 支 4 元钱的笔，还剩 1 元，则 4 元钱的笔少买 1 支，换成一支 5 元笔即可；若买完 x div 4 支 4 元钱的笔，还剩 2 元，则 4 元钱的笔少买 1 支，换成一支 6 元笔即可；若买完 x div 4 支 4 元钱的笔，还剩 3 元，则 4 元钱的笔少买 2 支，换成一支 5 元笔和一支 6 元笔即可。从以上对买笔方案的调整，可以看出笔的数目都是 x div 4，因此该方案的确为最优方案。源程序如下：

```
var a,b,c:integer;{a,b,c 分别表示在买笔方案中，6 元、5 元和 4 元钱笔的数目}
    x,y:integer;{x，y 分别表示剩余班费和买完最多的 4 元笔后剩的钱}
begin
  write( 'x=' );readln(x);  {输入 x}
  c:=x div 4;               {4 元笔最多买的数目}
  y:=x mod 4;               {求买完 c 支 4 元笔后剩余的钱数 y}
  case y of
    0 : begin  a:=0;b:=0; end;
    1 : begin  a:=0;b:=1;c:=c-1; end;
    2 : begin  a:=1;b:=0; c:=c-1;end;
    3 : begin  a:=1;b:=1; c:=c-2;end;
  end;
  writeln( 'a=' ,a,' b=' ,b,' c=' ,c);
end.
```

【上机练习 3.3】

1. 将字母 A、B、C 、D 或 a、b、c、d 转换成 1、2、3、4，其余的字符转换成 5。

2. 月收入 T 的所得税税率 R 如下：

T	R
T<800	0
1000>T>=800	5%
1500>T>=1000	10%
3000>T>=1500	15%
T>=3000	20%

分别用 IF 语句和 CASE 语句编写程序，输入月收入，输出所得税率、应缴所得税款以及扣除所得税后的实际收入。

第四章　循环结构程序设计

在实际应用中，会经常遇到许多有规律性的重复运算，这就需要掌握本章所介绍的循环结构程序设计。在 Pascal 语言中，循环结构程序通常由三种的循环语句来实现。它们分别为 FOR 循环、当循环和直到循环。通常将一组重复执行的语句称为循环体，而控制重复执行或终止执行由重复终止条件决定。重复语句是由循环体及重复终止条件两部分组成。

第一节　循环语句（FOR 语句）

一、for 语句的一般格式

for 〈控制变量〉:=〈表达式 1〉 to 〈表达式 2〉 do 〈语句〉;

for 〈控制变量〉:=〈表达式 1〉 downto 〈表达式 2〉 do 〈语句〉;

其中 for、to、downto 和 do 是 Pascal 保留字。表达式 1 与表达式 2 的值称为初值和终值。

循环的语句格式：FOR　变量名 := 初值　TO 终值　DO　语句;

[例] S：=0;

```
FOR  I ：= 1  TO  10  DO  S ：= S + I;
Writeln (‘S =’, S);
```

求 1+2+3+……+N 的和。如何编程呢？

二、For 语句执行过程

①先将初值赋给左边的变量（称为循环控制变量）;

②判断循环控制变量的值是否已“超过”终值，如已超过，则跳到步骤⑥;

③如果未超过终值，则执行 do 后面的那个语句（称为循环体）;

④循环变量递增（对 to）或递减（对 downt o）1 ;

⑤返回步骤②;

⑥循环结束，执行 for 循环下面的一个语句。

三、说明

①循环控制变量必须是顺序类型。可以是整型、字符型、枚举型等，但不能为实型。

②循环控制变量的值递增或递减的规律是：选用 to 则为递增；选用 downto 则递减。

③所谓循环控制变量的值“超过”终值，对递增型循环，“超过”指大于，对递减型循环，“超过”指小于。

④循环体可以是一个基本语句，也可以是一个复合语句。

⑤循环控制变量的初值和终值一经确定，循环次数就确定了。但是在循环体内对循环变量的值进行修改，常常会使得循环提前结束或进入死环。建议不要在循环体中随意修改控制变量的值。

⑥for 语句中的初值、终值都可以是顺序类型的常量、变量、表达式。

四、应用举例

例 1. 输出 1－100 之间的所有偶数。

```
var i:integer;
begin
  for i:=1 to 100 do
  if  i mod 2=0 then write(i:5);
end.
```

例 2. 求 N! ＝1＊2＊3＊…＊N ，这里 N 不大于 10。

分析：程序要先输入 N，然后从 1 累乘到 N。

程序如下：

```
var  n,i:integer;       {i 为循环变量}
       S:longint;       {s 作为累乘器}
begin
  write( ‘Enter n=’ );readln(n);{输入 n}
 s:=1;
 for i:=2 to n do       {从 2 到 n 累乘到 s 中}
  s:=s*i;
writeln(n,’ !=’ ,s);   {输出 n!的值}
end.
```

例 3、一个两位数 x，将它的个位数字与十位数字对调后得到一个新数 y，此时 y 恰好比 x 大 36，请编程求出所有这样的两位数。

分析：① 用 for 循环列举出所有的两位数，x 为循环变量；
② 用公式 a:= x div 10 分离出 x 的十位数字；
③ 用公式 b:= x mod 10 分离出 x 的个位数字；
④ 用公式 y:= b*10+a 合成新数 y；
⑤ 用式子 y-x=36 筛选出符合条件的数 x 并输出。

```
Program ex34;
 Var a,b,x,y:integer;
Begin
  For x := 10 to 99 do
  Begin
     a := x div 10;
     b := x mod 10;
     y := b*10+a;
     if y-x=36 then writeln(x);
   End;
   Readln;
End.
```

例 4：输入一个自然数，求这个自然数的所有约数之和。

分析：输入 X——>找出 X 的所有约数（从 1 到 X 逐个判断，看 X MOD Y 是否为 0），并且累加起来存在 S 中——>输出 S。

```
Program ex35;
 Var s,x,y:integer;
 BEGIN
   READLN (X); S：=0;
   FOR  Y：=1  TO  X   DO
        IF   X   MOD  Y = 0  THEN  S：=S+Y;
   WRITELN (S);
END.
```

例 5、把整数 3025 从中剪开分为 30 和 25 两个数，此时再将这两数之和平方，$(30+25)^2=3025$ 计算结果又等于原数。求所有符合这样条件的四位数。

分析：设符合条件的四位数为 N，它应当是一个完全平方数，用(a*a)表示。
① 为了确保 N=(a*a)在四位数（1000～9999）范围内，可确定 a 在 32～99 循环；
② 计算 N=a*a；将四位数 N 拆分为两个数 n1 和 n2；
③ 若满足条件(n1+n2)*(n1+n2)＝N 就输出 N 。

Pascal 程序:

```
Program  Exam35;
Var N, a,  x, n1, n2: Integer;
Begin
  for a:=32 to 99 do
  begin
    N:=a*a;
    n1:= N div 100;        {拆取四位数的前两位数}
    n2:= N-n1*100;         {拆取四位数的后两位数}
    X:=n1+n2;
    if  x*x=N  then  writeln (N);
  end;
  Readln
End.
```

【上机练习 4.1】

1、计算 n!，其中 n 由键盘输入。
2. 计算 100 之内所有的奇数之和。
3. 求菲波拉契数列 a0, a1, a2, ……a20。 a0=0, a1=1, a2=a1+a0, a3=a2+a1, ……,
$a_n=a_{n-1}+a_{n-2}$; 如 0, 1, 1, 2, 3, 5, 8, 13, 21, ……
4. 求 20 个数中的最大值和最小值。
5. 求 s=1+2+3+4+…+10
6. 求 s=1+1/2+1/3+…+1/100
7. 按字母表的顺序，从字母 A 到 Z 顺序打印输出。
8. 输入 10 个数，打印出最大和最小的数。
9. 编写一个评分程序，接受用户输入 10 个选手的得分(0-10 分)，然后去掉一个最高分和一个最低分，求出某选手的最后得分(平均分)。
10. 输入 30 个学生成绩，分别统计成绩在 85—100 分，60—85 分，60 分以下，各分数段中的人数。

第二节 当语句（ WHILE 语句）

一、WHILE 循环

对于 for 循环有时也称为计数循环，当循环次数未知，只能根据某一条件来决定是否进行循环时，用 while 语句或 repeat 语句实现循环要更方便。

while 语句的形式为：

while <布尔表达式> do <语句>;

其意义为：当布尔表达式的值为 true 时，执行 do 后面的语句。

while 语句的执行过程为：

①判断布尔表达式的值, 如果其值为真, 执行步骤 2, 否则执行步骤 4;

②执行循环体语句(do 后面的语句);

③返回步骤 1;

④结束循环, 执行 while 的下一个语句。

说明：这里 while 和 do 为保留字，while 语句的特点是先判断，后执行。 当布尔表达式成立时，重复执行 do 后面的语句(循环体)。

例 6、求恰好使 s=1+1/2+1/3+…+1/n 的值大于 10 时 n 的值。

分析：“恰好使 s 的值大于 10”意思是当表达式 s 的前 n-1 项的和小于或等于 10，而加上了第 n 项后 s 的值大于 10。从数学角度，我们很难计算这个 n 的值。故从第一项开始，当 s 的值小于或等于 10 时，就继续将下一项值累加起来。当 s 的值超过 10 时，最后一项的项数即为要求的 n。

程序如下：

```
var   s:real;
      n:integer;    {n 表示项数}
begin
  s:=0.0;n:=0;
  while s<=10 do    {当 s 的值还未超过 10 时}
    begin
      n:=n+1;       {项数加 1}
      s:=s+1/n;     {将下一项值累加到 s}
    end;
  writlen('n=',n);  {输出结果}
end.
```

例 7、求两个正整数 m 和 n 的最大公约数。

分析：求两个正整数的最大公约数采用的辗转相除法求解。以下是辗转的算法：

分别用 m, n, r 表示被除数、除数、余数。

①求 m/n 的余数 r.

②若 r=0, 则 n 为最大公约数. 若 r≠0, 执行第③步.

③将 n 的值放在 m 中,将 r 的值放在 n 中.
④返回重新执行第①步。

程序如下:

```
program ex4_4;
var m,n,a,b,r:integer;
begin
     write('Input m,n:');
     readln(m,n);
     a:=m;b:=n;r:=a mod b;
     while r<>0 do
     begin
       a:=b;b:=r;
       r:=a mod b;
     end;
     writeln('The greatest common divide is:',b:8);
end.
```

【上机练习 4.2】

1、用 WHILE 循环完成如下 3 题:
①求 s=1+2+3+4+…+10
②求 s=1+1/2+1/3+…+1/100
③求 π 的值。
已知 π/4=1 - 1/3 +1/5 - 1/7 + 1/9 - ……, 要求最后一项小于 10^{-6} 为止。
2、输入任一的自然数 A, B, 求 A , B 的最小公倍数。
3、Faibonacci 数列前几项为: 0,1,1,2,3,5,8,…,其规律是从第三项起, 每项均等于前两项之和。求前 30 项,并以每行 5 个数的格式输出。
4、小球从 100 高处自由落下,着地后又弹回高度的一半再落下。求第 20 次着地时, 小球共通过多少路程?
5、鸡兔同笼,头 30,脚 90, 求鸡兔各几只?

第三节 直到循环（REPEAT 语句）

用 while 语句可以实现“当型循环”，用 repeat-until 语句可以实现“直到型循环”。repeat-until 语句的含义是:“重复执行循环，直到指定的条件为真时为止”。

直到循环语句的一般形式:

```
Repeat
  <语句 1>;
  :
  <语句 n>;
until <布尔表达式>;
```

其中 Repeat、until 是 Pascal 保留字，repeat 与 until 之间的所有语句称为循环体。

说明:

①repeat 语句的特点是：先执行循环，后判断结束条件，因而至少要执行一次循环体。

②repeat-until 是一个整体，它是一个（构造型）语句，不要误认为 repeat 是一个语句，until 是另一个语句。

③repeat 语句在布尔表达式的值为真时不再执行循环体，且循环体可以是若干个语句，不需用 begin 和 end 把它们包起来， repeat 和 until 已经起了 begin 和 end 的作用。while 循环和 repeat 循环是可以相互转化的。

例 8、校体操队到操场集合,排成每行 2 人,最后多出 1 人;排成每行 3 人,也多出 1 人;分别按每行排 4,5,6 人,都多出 1 人;当排成每行 7 人时,正好不多。求校体操队至少是多少人?

分析：①设校体操队为 X 人,根据题意 X 应是 7 的倍数,因此 X 的初值为 7,以后用 inc(x,7)改变 X 值;
②为了控制循环, 用逻辑变量 yes 为真(True) 使循环结束;

③如果诸条件中有一个不满足， yes 的值就会为假(false)，就继续循环。

```
program  Exam311;
var  x: integer;  yes : boolean;
begin
   x:=0;
   repeat
     yes  :=true;  inc(x,7);
     if x mod 2 < > 1 then yes:=false;
     if x mod 3 < > 1 then yes:=false;
     if x mod 4 < > 1 then yes:=false;
     if x mod 5 < > 1 then yes:=false;
     if x mod 6 < > 1 then yes:=false;
   until yes;                          {直到 yes 的值为真 }
   writeln('All =', x) ;  readln
end.
```

程序中对每个 X 值，都先给 Yes 赋真值，只有在循环体各句对 X 进行判断时，都得到“通过”（此处不赋假值）才能保持真值。

例 9、求 1992 个 1992 的乘积的末两位数是多少？

分析：积的个位与十位数只与被乘数与乘数的个位与十位数字有关，所以本题相当于求 1992 个 92 相乘，而且本次的乘积主下一次相乘的被乘数，因此也只需取末两位参与运算就可以了。

Pascal 程序：

```
Program ex313;
var a,t : integer;
Begin
  a := 1;
  t := 0;
  repeat
    t := t+1;
    a := (a*92) mod 100;
  until t=1992;
  writeln(a);
  Readln;
End.
```

以上我们已介绍了三种循环语句。一般说来，用 for 循环比较简明，只要能用 for 循环，就尽量作用 for 循环。只在无法使用 for 循环时才用 while 循环和 repeat-until 循环， 而且 while 循环和 repeat-until 循环是可以互相替代的。for 循环在大多数场合也能用 whiel 和 repeat-until 循环来代替。一般 for 循环用于有确定次数循环，而 while 和 repeat-until 循环用于未确定循环次数的循环。

【上机练习 4.3】

1、用 REPEAT 循环完成如下 3 题：

①求 s=1+2+3+4+…+10

②求 s=1+1/2+1/3+…+1/100

③求 π 的值。

已知 π/4=1 - 1/3 +1/5 - 1/7 + 1/9 - ……，要求最后一项小于 10^{-6} 为止。

2. 读一组实数,遇零终止,打印其中正、负数的个数及各自的总和。

3. 计算 sin（x）=$X-X^3/3!+X^5/5!-X^7/7!+$……直到最后一项绝对值小于 10^{-7} 时停止计算，x 由键盘输入 。

4. 用辗转相除法求两个自然数的最大公约数。

5. 找出被 2、3、5 除时余数为 1 的最小的十个数。

6. 将一根长为 369cm 的钢管截成长为 69cm 和 39cm 两种规格的短料。在这两种规格的短料至少各截一根的前提下，如何截才能余料最少。

第四节　多重循环结构

当一个循环的循环体中又包含循环结构程序时，我们就称之为循环嵌套。

例 10、求 1!+2!+…+10!的值。

分析：这个问题是求 10 自然数的阶乘之和，可以用 for 循环来实现。程序结构如下：

```
for n:=1 to 10 do
 begin
   ①N!的值 →t
   ②累加 N!的值 t
 end
```

显然,通过 10 次的循环可求出 1!,2!…,10!,并同时累加起来，可求得 S 的值。而求 T=N!,又可以用一个 for 循环来实现:

```
 t=1;
 for j:=1 to n do
    t:=t*j;
```

因此,整个程序为:

```
program ex4_5;
var  t,s:real; i,j,n:integer;
begin
  S:=0;
  for n:=1 to 10 do
  begin
    t:=1;
    for j:=1 to n do
       t:=t*j;
    S:=S+t;
  end;
  writeln( 's=' ,s:0:0);
end.
```

以上的程序是一个二重的 for 循环嵌套。这是比较好想的方法，但实际上对于求 n!，我们可以根据求出的(n-1)!乘上 n 即可得到，而无需重新从 1 再累乘到 n。程序可改为：

```
program ex4_5;
var  t,s:real;  i,j,n:integer;
begin
  S:=0;t:=1;
  for n:=1 to 10 do
  begin
    t:=t*n;S:=S+t;
  end;
  writeln( 's=' ,s:0:0);
end.
```

显然第二个程序的效率要比第一个高得多。第一程序要进行 1+2+…+10=55 次循环，而第二程序进行 10 次循环。如题目中求的是 1！＋2！＋…＋1000!，则两个程序的效率区别更明显。

例 11、一个炊事员上街采购,用 500 元钱买了 90 只鸡，其中母鸡一只 15 元,公鸡一只 10 元,小鸡一只 5 元,正好把钱买完。问母鸡、公鸡、小鸡各买多少只？

分析：设母鸡 I 只,公鸡 J 只,则小鸡为 90-I-J 只,则 15*I+ 10* J+(90-I-J)*5=500,显然一个方程求两个未知数是不能直接求解。必须组合出所有可能的 i,j 值，看是否满足条件。这里 I 的值可以是 0 到 33，J 的值可以 0 到 50。源程序如下：

```
 programr ex4_6;
 var i,j,k:integer;
 begin
   for i:=1 to 33 do
   for j:=1 to 50 do
```

```
        begin
          k:=90-i-j;
          if 15*i+10*j+5*k=500 then writeln(i:5,j:5,k:5);
        end;
    end.
```

例 12、求 100－200 之间的所有素数。

分析：我们可对 100－200 之间的每一整数进行判断，判断它是否为素数，是则输出。而对于任意整数 i，根据素数定义，我们从 2 开始，到 $\sqrt{i}$，找 i 的第一个约数。若找到第一个约数，则 i 必然不是素数。否则 i 为素数。源程序如下：

```
programr ex4_7;
  var   I,x :integer;
   begin
    for i:=100 to 200 do
      begin
        x:=2;
        while (x<=trunc(sqrt(i)))and(i mod x<>0)do
          begin
            x:=x+1;
          end;
        if x>trunc(sqrt(i)) then write(i:8);
      end;
  end.
```

【上机练习 4.4】

1. 求 s=1!+2!+3!+…+10!
2. 求 s=1+1/2!+1/3!+…+1/10!
3. 求 $s=1^1+2^2+3^3+..+N^N$
4. 把一张一元钞票换成一分，二分和五分的硬币，每种至少一枚。问有哪几种换法？
5. 输入一个整数，若是素数，输出“YES”，否则输出“NO”
6. 任给一个自然数 n，求出这个自然数不同因数的个数。
   如：n=6 时，因为 1，2，3，6 这四个数均是 6 的因数，故输出为 total=4。
7. 输入二个正整数，求出它们的最大公约数和最小公倍数。
8. 输入一列图形（字母金字塔）

```
                a
              a   b
            a   b   c
          .           .
        a  b  c ……  y  z
```

9. 1-100 之间的所有素数（素数是大于 1，且除 1 和它本身外，不能被任何其它整数所整除的整数）。(4.28)
10. 哥德巴赫猜想（任何充分大的偶数都可由两个素数之和表示）。将 4-100 中的所有偶数分别用两个素数之和表示。输出为：

```
4=2+2
6=3+3
….
100=3+97
```

11. 某人想将手中的一张面值 100 元的人民币换成 10 元、5 元、2 元和 1 元面值的票子。要求换正好 40 张，且每种票子至少一张。问：有几种换法？应适当考虑减少重复次数。
12. 百鸡问题：一只公鸡值 5 元，一只母鸡值 3 元，而 1 元可买 3 只小鸡。现有 100 元钱，想买 100 只鸡。问可买公鸡、母鸡、小鸡各几只？
13、编写一程序，验证角谷猜想。所谓的角谷猜想是：“对于任意大于 1 的自然数 n，若 n 为奇数，则将 n 变为 3*n+1，否则将 n 变为 n 的一半。经过若干次这样的变换，一定会使 n 变为 1。”
14. 有一堆 100 多个的零件，若三个三个数，剩二个；若五个五个数，剩三个；若七个七个数，剩五个。请你编一个程序计算出这堆零件至少是多少个？

第五章 枚举和子界类型

在前面几章中我们用到了整型、实型、布尔型、字符型的数据。以上数据类型是由 pascal 规定的标准数据类型，只要写 integer,real,boolean, char, pascal 编译系统就能识别并按这些类型来处理。pascal 还允许用户定义一些类型，这是其它一些语言所没有的，这就使得 pascal 使用灵活、功能丰富。

第一节 枚举类型

一、枚举类型

随着计算机的不断普及，程序不仅只用于数值计算，还更广泛地用于处理非数值的数据。例如，性别、月份、星期几、颜色、单位名、学历、职业等，都不是数值数据。

在其它程序设计语言中，一般用一个数值来代表某一状态，这种处理方法不直观，易读性差。如果能在程序中用自然语言中有相应含义的单词来代表某一状态，则程序就很容易阅读和理解。也就是说，事先考虑到某一变量可能取的值，尽量用自然语言中含义清楚的单词来表示它的每一个值，这种方法称为枚举方法，用这种方法定义的类型称枚举类型。

枚举类型是一种很有实用价值的数据类型，它是 pascal 一项重要创新。

（一）枚举类型的定义

枚举类型是一种自定义类型，要使用枚举类型当然也要先说明枚举类型。

枚举类型的一般格式:

(标识符 1,标识符 2,…,标识符 n)

说明：①括号中的每一个标识符都称为枚举元素或枚举常量。

②定义枚举类型时列出的所有枚举元素构成了这种枚举类型的值域（取值范围），也就是说，该类型的变量所有可能的取值都列出了。

例如，下列类型定义是合法的：

```
type  days=(sun,mon,tue,wed,thu,fri,sat);
      colors=(red,yellow,blue,white,black,green);
```

而下列类型定义是错误的(因为枚举元素非标识符):

```
type  colortype=('red','yellow','blue','white');
      numbers=(1,3,5,7,9);
      ty=(for,do,while);
```

（二）枚举类型变量

定义了枚举类型，就可以把某些变量说明成该类型。如：

```
var holiday,workday:day;
    incolor:colors;
```

也可以把变量的说明与类型的定义合并在一起,如:

```
var holiday,workday:(sun,mon,tue,wed,thu,fri,sat);
    incolor:(red,yellow,blue,white,black,green);
```

（三）枚举类型的性质

1.枚举类型属于顺序类型

根据定义类型时各枚举元素的排列顺序确定它们的序号，第一个枚举元素的序号为 0。例如：设有定义：

```
type days=(sun,mon,tue,wed,thu,fri,sat);
```

则:

ord(sun)=0,ord(mon)=1,ord(sat)=6;succ(sun)=mon,succ(mon)=tue

succ(fri)=sat;pred(mon)=sun,pred(tue)=mon,pred(sat)=fri。

应注意的是：枚举类型中的第一个元素无前趋，最后一个元素无后继。

2.对枚举类型只能进行赋值运算和关系运算

一旦定义了枚举类型及这种类型的变量，则在语句部分只能对枚举类型变量赋值，或进行关系运算，不能进行算术运算和逻辑运算。

在枚举元素比较时，实际上是对其序号的比较。当然，赋值或比较时，应注意类型一致。例如，设程

序有如下说明：

```
type days=(sun,mon,tue,wed,thu,fri,sat);
     colors=(red,yellow,blue,white,black,green);
var color:colors;
     weekday:days;
```

则下列比较或语句是合法的:

```
weekday:=mon;
if weekday=sun then write('rest');
weekday<>sun
```

而下面比较或语句是不合法的:

```
mon:=weekday;
mon:=1;
if weekday=sun or sat then write('rest');
sun>red
weekday<>color
```

3.枚举变量的值只能用赋值语句来获得

也就是说，不能用 read(或 readln)读一个枚举型的值。同样，也不能用 write(或 writeln)输出一个枚举型的值。如 write(red)是非法的，会发生编译错误。千万不要误认为，该语句的结果是输出“red”三个字符。

但对枚举数据的输入与输出可通过间接方式进行。输入时，一般可输入一个代码，通过程序进行转换，输出时，也只是打印出与枚举元素相对应的字符串。这在后面的例题中将有使用示例。

4.同一个枚举元素不能出现在两个或两个以上的枚举类型定义中。

如:

```
type color1=(red,yellow,white);
     color2=(blue,red,black);
```

是不允许的，因为 red 属于两个枚举类型。

四、枚举类型应用举例

例 1：现有四种水果：苹果、橘子、香蕉、菠萝。从四种水果中任取三种，不能重复和不计顺序。编程列出一切组合。

（设四种水果：苹果-aplle；橘子-orange；香蕉-banana；菠萝-pineapple）

```
program xx;（解法一）
type fruit=(apple,orange,banana,pineapple);
var  i,j:fruit;
begin
  for i:=apple to pineapple do
  begin
  for j:=apple to pineapple do
    if i<>j then
    begin
     case j of
        apple:write('apple':12);
        orange:write('orange':12);
        banana:write('banana':12);
        pineapple:write('pineapple':12);
     end;
    end;
  writeln;
  end;
  readln;
end.
program fruit2; （解法二）
var    i,j,k,p,l:integer;
begin
```

```
  for i:=1 to 2 do
   for j:=i+1 to 3 do
    for k:=j+1 to 4 do
     begin
      for l:=1 to 3 do
       begin
        case l of
            1:p:=i;
            2:p:=j;
            3:p:=k;
        end;
        case p of
            1:write('apple':12);
            2:write('orange':12);
            3:write('banana':12);
            4:write('pineapple':12);
        end;
       end;
      writeln;
     end;
    end.
```

例 2、一周七天用 sun,mon,tue,wed,thu,fri,sat 表示， 要求利用枚举类型编程：当输入星期几的数字,能输出它的后一天是星期几(也用英文表示)。

源程序如下：

```
  program ex6_1;
  type week=(sun,mon,tue,wed,thu,fri,sat);
  var
      i           :    integer;
      day,sucday  :    week;
  begin
    write('What date is it');readln(i);
    case i of                          {根据输入 i 转换成枚举型}
      1:day:=mon;
      2:day:=tue;
      3:day:=wed;
      4:day:=thu;
      5:day:=fri;
      6:day:=sat;
      7:day:=sun;
    end;
    if (day=sat) then sucday:=sun  else sucday:=succ(day); {计算明天 sucday}
    write('The next day is ');      {输出明天是星期几}
    case sucday of
      sun:writeln('sunday');
      mon:writeln('monday');
      tue:writeln('tuesday');
      wed:writeln('wednesay');
      thu:writeln('thursday');
      fri:writeln('friday');
      sat:writeln('saturday');
    end;
  end.
```

评注：程序中变量 day、sucday 分别表示今天、明天。

第二节　子界类型

如果我们定义一个变量 i 为 integer 类型，那么 i 的值在微型机系统的 pascal 中,使用 2 字节的定义表示法,取值范围为-32768～32767。而事实上，每个程序中所用的变量的值都有一个确定的范围。

例如，人的年龄一般不超过 150，一个班级的学生不超过 100 人，一年中的月数不超过 12，一月中的天数不超过 31，等等。

如果我们能在程序中对所用的变量的值域作具体规定的话，就便于检查出那些不合法的数据，这就能更好地保证程序运行的正确性。而且在一定程度上还会节省内存空间。

子界类型就很好解决如上问题。此外，在数组的定义中，常用到子界类型，以规定数组下标的范围，上一章有关数组知识中我们已用到。

（一）子界类型定义

子界类型的一般格式：

<常量 1>..<常量 2>

说明：①其中常量 1 称为子界的下界,常量 2 称为子界的上界。

②下界和上界必须是同一顺序类型（该类型称为子界类型的基类型），且上界的序号必须大于下界的序号。例如，下列说明：

```
type age=0.5..150;
     letter=0..'z';
     let1='z'..'a';
```

都是错误的。

③可以直接在变量说明中定义子界类型。如：

```
type letter='a'..'d';
var ch1,ch2:letter;
```

可以合并成：`var ch1,ch2:'a'..'d';`

当然，将类型定义和变量定义分开，则更为清晰。

（二）子界类型数据的运算规则

1.凡可使用基类型的运算规则同样适用该类型的子界类型。

例如，可以使用整型变量的地方，也可以使用以整型为基类型的子界类型数据。

2.对基类型的运算规则同样适用于该类型的子界类型。

例如，div，mod 要求参加运算的数据为整，因而也可以为整型的任何子界类型数据。

3. 基类型相同的不同子界类型数据可以进行混合运算。

例如：设有如下说明：

```
type  a=1..100;  b=1.1000; c=1..500;
var   x:a;  y:b;  t:c;
      z:integer;
```

则下列语句也是合法的：

```
Z:=Sqr(x)+y+t;
```

下列语句:

```
  t:=x+y+z;
```

当 X+Y+Z 的值在 1～500 范围内时是合法的，否则会出错。

（三）子界类型应用举例

例 3、 利用子界类型作为情况语句标号，编一个对数字， 大小写字母和特殊字符进行判别的程序。

源程序如下：

```
program cas;
 var c:char;
 begin
   readln(c);
   case c of
     '0'..'9':writeln('digits');
     'A'..'Z':writeln('UPPER-CASELETTERS');
```

```
        'a'..'z':writeln('lower-caseletters');
        esle writeln('special charactors');
    end;
end.
```

例 4、使用子界型情况语句，当输入月、日、年(10 30 1986)，输出 30 Oct 1986。

源程序如下:

```
program ex6_3;
  var month:1..12;  date:1..31;  year:1900..1999;
  begin
    write('Enter date(mm-dd-yy):');
    readln(month,date,year);
    write(date);
    case month of
      1:write('Jan':5);
      2:write('Feb':5);
      3:write('Mar':5);
      4:write('Apr':5);
      5:write('May':5);
      6:write('Jun':5);
      7:write('Jul':5);
      8:write('Aug':5);
      9:write('Sep':5);
     10:write('Oct':5);
     11:write('Nov':5);
     12:write('Dec':5);
    end;
    writeln(year:7);
  end.
```

枚举类型和子界类型均是顺序类型，在前面一章数组的定义时，实际上我们已经用到了子界类型，数组中的下标类型确切地讲可以是和枚举类型或子界类型，大多数情况下用子界类型。

如有如下说明：

```
type color=(red,yellow,blue,white,black);
var   a:array[color]of integer;
      b:array[1..100]of color;
```

都是允许的。

【上机练习 5.1】

1、变量 s 已定义如下：

```
var s(knife,rule,pen,rubber)
```

且 s 中已有值，试写一程序，输出 s 中的值。

2、定义枚举类型 monthtype 表示十二个月，输入 1-12 中的某一个数，输出对应月份的英文缩写和表示下一个月的数字。如

输入：6

输出：jun next month: 7

3、由五个字符组成一个字符串，规定前四个字符为小写字母，第五个字符为数字，问有多少种排列方法。

4、类型定义 type ren=' A' ..' F'

用 A 至 F 表示 6 个人，输出 6 个人相互握手的各种情况，并统计握手的次数。

5、从红(red)、黄(yellow)、兰(blue)、白(white)、黑(black)五种颜色的球中，任取三种不同颜色的球，求所有可能的取法？

6、一家水果店出售 4 种水果，每千克价格分别是：苹果 1.15 元，桔子 1.20 元，香蕉 0.95 元，菠萝 0.85 元。编一程序使售货员主要从键盘上打入货品的代码及重量，计算机将显示货品名、单价、重量及总价。货品代码为苹果 1，桔子 2，香蕉 3，菠萝 4。

第六章　数组

第一节　一维数组

一、为什么要使用数组

例 1　输入 50 个学生的某门课程的成绩，打印出低于平均分的同学号数与成绩。

【问题分析】

在解决这个问题时，虽然可以通过读入一个数就累加一个数的办法来求学生的总分，进而求出平均分。但因为只有读入最后一个学生的分数以后才能求得平均分，且要打印出低于平均分的同学，故必须把 50 个学生的成绩都保留下来， 然后逐个和平均分比较，把高于平均分的成绩打印出来。如果，用简单变量 a1，a2,…，a50 存放这些数据，可想而知程序要很长且繁。要想如数学中使用下标变量 a_i 形式表示这 50 个数，则可以引入下标变量 a[i]。这样问题的程序可写为：

```
tot:=0;              {tot 表示总分}
for i:=1 to 50 do  {循环读入每一个学生的成绩，并累加它到总分}
  begin
    read(a[i]);
    tot:=tot+a[i];
  end;
ave:=tot/50;         {计算平均分}
for i:=1 to 50 do
 if a[i]<ave then writeln( 'No.' ,i,'   ',a[i]); {如果第 i 个同学成绩小于平均分，则输出}
```

而要在程序中使用下标变量，则必须先说明这些下标变量的整体－数组，即数组是若干个同名（如上面的下标变量的名字都为 a）下标变量的集合。

二、一维数组

当数组中每个元素只带有一个下标时，我们称这样的数组为一维数组。

1、一维数组的定义

（1）类型定义

要使用数组类型等构造类型以及第 5 章学习的自定义类型(枚举类型与子界类型)，应在说明部分进行类型说明。 这样定义的数据类型适用整个程序。

类型定义一般格式为：

```
type
  <标识符 1>=<类型 1>;
  <标识符 2>=<类型 2>;
  :
  <标识符 n>=<类型 n>;
```

其中 type 是 Pascal 保留字,表示开始一个类型定义段。在其后可以定义若干个数据类型定义。<标识符>是为定义的类型取的名字，称它为类型标识符。

类型定义后，也就确定了该类型数据取值的范围，以及数据所能执行的运算。

（2）一维数组类型的定义

一维数组类型的一般格式：

```
array [下标 1..下标 2] of <基类型>;
```

说明：其中 array 和 of 是 pascal 保留字。下标 1 和下标 2 是同一顺序类型，且下标 2 的序号大于下标 1 的序号。它给出了数组中每个元素(下标变量) 允许使用的下标类型，也决定了数组中元素的个数。基类型是指数组元素的类型，它可以是任何类型，同一个数组中的元素具有相同类型。因此我们可以说，数组是由固定数量的相同类型的元素组成的。

再次提请注意：类型和变量是两个不同概念，不能混淆。就数组而言，程序的执行部分使用的不是数组类型（标识符）而是数组变量（标识符）。

一般在定义数组类型标识符后定义相应的数组变量，如：

```
type arraytype=array [1..8] of integer;
var  a1,a2:arraytype;
```

其中 arraytype 为一个类型标识符,表示一个下标值可以是 1 到 8,数组元素类型为整型的一维数组；

而 a1,a2 则是这种类型的数组变量。

也可以将其合并起来：

```
var a1,a2:array [1..8] of integer;
```

当在说明部分定义了一个数组变量之后,pascal 编译程序为所定义的数组在内存空间开辟一串连续的存储单元。

例如，设程序中有如下说明：

```
type rowtype=array [1..8] of integer;
     coltype=array ['a'..'e'] of integer;
var a:rowtype;b:coltype;
```

2、一维数组的引用

当定义了一个数组，则数组中的各个元素就共用一个数组名(即该数组变量名)，它们之间是通过下标不同以示区别的。 对数组的操作归根到底就是对数组元素的操作。一维数组元素的引用格式为：

数组名[下标表达式]

说明：①下标表达式值的类型，必须与数组类型定义中下标类型完全一致,并且不允许超越所定义的下标下界和上界。

②数组是一个整体，数组名是一个整体的标识，要对数组进行操作，必须对其元素操作。数组元素可以象同类型的普通变量那样作用。如:a[3]:=34;是对数组 a 中第三个下标变量赋以 34 的值。read(a[4]);是从键盘读入一个数到数组 a 第 4 个元素中去。

特殊地,如果两个数组类型一致,它们之间可以整个数组元素进行传送。如：

```
var a,b,c:array[1..100] of integer;
begin
  c:=a;a:=b;b:=c;
end.
```

在上程序中，a,b,c 三个数组类型完全一致， 它们之间可以实现整数组传送，例子中，先将 a 数组所有元素的值依次传送给数组 c，同样 b 数组传给 a，数组 c 又传送给 b，上程序段实际上实现了 a,b 两个数组所有元素的交换。

对于一维数组的输入与输出，都只能对其中的元素逐个的输入与输出。在下面的应用示例中将详细介绍。

三、一维数组应用示例

一维数组元素的输入

不能整个数组输入，只能逐个元素赋值，A [I] : = X ;

一般用 FOR 循环做。如：

```
FOR  I : =  1  TO  7  DO  READ ( A [ I ] );
```

一维数组元素的输出

不能整个数组一起输出，只能逐个元素输出，WRITE (A [I]);

一般用 FOR 循环做。如：

```
FOR  I : =  1  TO  7  DO  WRITE ( A [ I ] );
```

例 2　输入 50 个数,要求程序按输入时的逆序把这 50 个数打印出来。也就是说，请你按输入相反顺序打印这 50 个数。

【问题分析】我们可定义一个数组 a 用以存放输入的 50 个数，然后将数组 a 内容逆序输出。

```
program ex5_1;
  type arr=array[1..50]of integer;   {说明一数组类型 arr}
  var a:arr;i:integer;
  begin
    writeln('Enter 50 integer:');
    for i:=1 to 50 do read(a[i]);    {从键盘上输入 50 个整数}
    readln;
    for i:=50 downto 1 do            {逆序输出这 50 个数}
      write(a[i]:10);
  end.
```

例 3、将 a 数组中第一个元素移到最后数组末尾,其余数据依次往前平移一个位置。

【问题分析】

为完成题目所要求的操作，其算法应该包括以下几个主要步骤：

①把第一个元素的值取出入在一个临时单元 temp 中；

②通过 a[2]→a[1], a[3]→a[2], a[4]→a[3], ……, a[n]→a[n-1],实现其余元素前移

③将 temp 值送入 a[n].

```
program p6-4;
const n=10;
var a:array[1..n] of integer;
    i:integer; temp:integer;
begin
    writeln( 'reak',n,'datas');
    for i:=1 to n do read(a[i]);
    temp:=a[1];
    for i:=1 to n-1 do a[i]:=a[i+1];
    a[n]:=temp;
    writeln( 'Result:');
    for i:=1 to n do write(a[i]:3);
end.
```

运行结果 :

```
read 10 datas:
•  1 2 3 4 5 6 7 8 9 10
Result:
•  2 3 4 5 6 7 8 9 10 1
```

例 4、输入十个正整数,把这十个数按由大到小的顺序排列。(选择排序)

将数据按一定顺序排列称为排序，排序的算法有很多，其中选择排序是一种较简单的方法。

【问题分析】

要把十个数按从大到小顺序排列，则排完后，第一个数最大，第二个数次大，……。因此，我们第一步可将第一个数与其后的各个数依次比较，若发现，比它大的，则与之交换，比较结束后，则第一个数已是最大的数（最大的泡往上冒）。同理，第二步，将第二个数与其后各个数再依次比较，又可得出次大的数。如此方法进行比较，最后一次，将第九个数与第十个数比较，以决定次大的数。于是十个数的顺序排列结束。

如对 5 个进行排序，这个五个数分别为 8 2 9 10 5。按选择排序方法，过程如下：

```
初始数据  : 8  2  9  10  5
第一次排序: 8  2  9  10  5
            9  2  8  10  5
            10  2  8  9  5
            10  2  8  9  5
第二次排序: 10  8  2  9  5
            10  9  2  8  5
            10  9  2  8  5
第三次排序: 10  9  8  2  5
            10  9  8  2  5
第四次排序: 10  9  8  5  2
```

对于十个数，则排序要进行 9 次。源程序如下：

```
program ex5_2;
 var a:array[1..10]of integer;
      i,j,t:integer;
  begin
    writeln('Input 10 integers:');
    for i:=1 to 10 do read(a[i]); {读入 10 个初始数据}
    readln;
```

```
    for i:=1 to 9 do                {进行 9 次排序}
      begin
        for j:=i+1 to 10 do         {将第 i 个数与其后所有数比较}
          if a[i]<a[j] then         {若有比 a[i]大,则与之交换}
             begin
               t:=a[i];a[i]:=a[j];a[j]:=t;
             end;
      end;
      for i:=1 to 10  do write(a[i]:5);
  end.
```

例 5、编程输入十个正整数，然后自动按从大到小的顺序输出。（冒泡排序）

【问题分析】

①用循环把十个数输入到A数组中；

②从A[1]到A[10]，相邻的两个数两两相比较，即：

A[1]与A[2]比，A[2]与A[3]比，……A[9]与A[10]比。

只需知道两个数中的前面那元素的标号，就能进行与后一个序号元素（相邻数）比较，可写成通用形式A[i]与A[i +1]比较，那么，比较的次数又可用1～(n - i)循环进行控制（即循环次数与两两相比较时前面那个元素序号有关）；

③在每次的比较中，若较大的数在后面，就把前后两个对换，把较大的数调到前面，否则不需调换位置。

下面例举5个数来说明两两相比较和交换位置的具体情形：

5	6	4	3	7	5和6比较，交换位置，排成下行的顺序；
6	5	4	3	7	5和4比较，不交换，维持同样的顺序；
6	5	4	3	7	4和3比较，不交换，顺序不变
6	5	4	3	7	3和7比较，交换位置，排成下行的顺序；
6	5	4	7	3	经过（1～（5-1））次比较后，将3调到了末尾。

经过第一轮的1～ (N-1)次比较，就能把十个数中的最小数调到最末尾位置，第二轮比较1～ (N-2)次进行同样处理，又把这一轮所比较的“最小数”调到所比较范围的“最末尾”位置；……；每进行一轮两两比较后，其下一轮的比较范围就减少一个。最后一轮仅有一次比较。在比较过程中，每次都有一个“最小数”往下“掉”，用这种方法排列顺序，常被称之为“冒泡法”排序。

```
Program  Exam52;
const  N=10;
Var a: array[1..N] of integer;                    {定义数组}
    i,j: integer;
procedure Swap(Var x,y: integer);                 {交换两数位置的过程}
Var t:integer;
begin
 t:=x;  x:=y;  y:=t
end;
 Begin
   for i:=1 to N do                               {输入十个数}
     begin
        Readln(a[ i ])
     end;
   for j:=1 to N-1 do                             {冒泡法排序}
     for i:=1 to N-j do                           {两两相比较}
        if a[ i ] < a[i+1] then swap(a[ i ], a[i+1]);  {比较与交换}
   for i:=1 to N do                               {输出排序后的十个数}
     write(a[ i ]:6);
   Readln
 end.
```

例6、用筛法求出100以内的全部素数，并按每行五个数显示。

【问题分析】

⑴ 把2到100的自然数放入a[2]到a[100]中（所放入的数与下标号相同）；

⑵ 在数组元素中，以下标为序，按顺序找到未曾找过的最小素数minp,和它的位置p（即下标号）；

⑶ 从p+1开始，把凡是能被minp整除的各元素值从a数组中划去（筛掉），也就是给该元素值置 0；

⑷ 让p=p+1，重复执行第②、③步骤，直到minp>Trunc(sqrt(N)) 为止；

⑸ 打印输出a数组中留下来、未被筛掉的各元素值，并按每行五个数显示。

用筛法求素数的过程示意如下（图中用下划线作删去标志）：

① 2 3 4 5 6 7 8 9 10 11 12 13 14 15…98 99 100　　{置数}

② 2 3 4 5 6 7 8 9 10 11 12 13 14 15…98 99 100　　{筛去被2整除的数 }

③ 2 3 4 5 6 7 8 9 10 11 12 13 14 15…98 99 100　　{筛去被3整除的数 }

……

2 3 4 5 6 7 8 9 10 11 12 13 14 15…98 99 100　　{筛去被整除的数 }

```
Program Exam53;
const  N=100;
type   xx=1 .. N;                                  {自定义子界类型xx（类型名）}
Var a: array[xx] of boolean; i,j: integer;
Begin
  Fillchar(a,sizeof(a),true);
  a[1] := False;
  for i:=2 to Trunc(sqrt(N)) do
    if a[I] then
        for j := 2 to N div I do
            a[I*j]:= False;
  t:=0;
  for i:=2 to N do
    if a[i] then
      Begin
        write(a[ i ]:5);  inc(t);
        if t mod 5=0 then writeln
      end;
End.
```

【上机练习 6.1】

1、国际象棋盘中，第 1 格放 1 粒米，第 2 格放 2 粒米，第 3 格放 4 粒米，第 4 格放 8 粒米，第 5 格放 16 粒米,..... 问：16 个格子总共可以放多少粒米？

分析：第 i 个格子可放多少粒米：2^{i-1}

2、输出斐波列契数列的前 N 项（5 个 1 行）

0 1 1 2 3 5 8 13 21 .....

3、输入 N 个整数，找出最大数所在位置，并将它与第一个数对调位置。

方法：“比武招亲”、“打擂台”

4、插入 1 个整数在 1 个有序数组中（从小到大，整型数据），要求插入后仍有序。

5、将一个数组中的所有元素倒序存放 。

分析：A[1]←→A[N]　A[2] ←→A[N-1]…… A[I] ←→A[J]

I 从 1 开始，每交换 1 次，I 加 1；直到 I = N　DIV 2

6、删除数组中的某元素，且右边的元素都向左平移一格。

7、查找数组 A 中是否有等于 NUM 这个数，有则返回这个数在数组中的位置；没有，则返回 0。（假设 A 中有 N 个互异的整数）

8、有 N 个数存放于数组 A 中，将其按照从小到大的顺序重新排列。

选择排序法：用“打擂台”法将最小的 1 个数找出来放在数组的最前面。然后在剩下的 N-1 个数中重复做上面的操作……..., 一共要 N - 1 趟。

第二节　多维数组

一、多维数组的定义

当一维数组元素的类型也是一维数组时，便构成了二维数组。二维数组定义的一般格式：

array[下标类型 1] of array[下标类型 2] of 元素类型;

但我们一般这样定义二维数组：

array[下标类型 1，下标类型 2] of 元素类型;

说明：其中两个下标类型与一维数组定义一样，可以看成“下界 1..上界 1”和“下界 2..上界 2”，给出二维数组中每个元素(双下标变量)可以使用下标值的范围。of 后面的元素类型就是基类型。

一般地，n 维数组的格式为：

array[下标类型 1，下标类型 2，…，下标类型 n] of 元素类型;

其中，下标类型的个数即数组的维数，且说明了每个下标的类型及取值范围。

二、多维数组元素的引用

多维数组的数组元素引用与一维数组元素引用类似,区别在于多维数组元素的引用必须给出多个下标。引用的格式为：

<数组名>[下标 1,下标 2,…,下标 n]

说明：显然，每个下标表达式的类型应与对应的下标类型一致，且取值不超出下标类型所指定的范围。

例如,设有说明:

```
type matrix=array[1..5,1..4]of integer;
var a:matrix;
```

则表示 a 是二维数组，共有 5*4=20 个元素，它们是：

a [1，1] a [1，2] a [1，3] a [1，4]
a [2，1] a [2，2] a [2，3] a [2，4]
a [3，1] a [3，2] a [3，3] a [3，4]
a [4，1] a [4，2] a [4，3] a [4，4]
a [5，1] a [5，2] a [5，3] a [5，4]

因此可以看成一个矩阵，a [4，2] 即表示第 4 行、第 2 列的元素。由于计算机的存储器是一维的，要把二维数组的元素存放到存储器中，pascal 是按行（第一个下标）的次序存放，即按 a [1，1] a [1，2] a [1，3] a [1，4] a [2，1] …，a [5，4] 的次序存放于存储器中某一组连续的存储单元之内。

对于整个二维数组的元素引用时,大多采用二重循环来实现。如：给如上说明的二维数组 a 进行赋值:a[i,j]=i*j。

```
for i:=1 to 5 do
   for j:=1 to 4 do
      a[i,j]:=i*j;
```

对二维数组的输入与输出也同样可用二重循环来实现:

```
for i:=1 to 5 do
begin
   for j:=1 to 4 do  read(a[i,j]);
end;
for i:=1 to 5 do
   begin
      for j:=1 to 4 do  write(a[i,j]:5);
   end;
```

三、多维数组的应用示例

例 7、设有一程序:

```
program ex5_3;
const n=3;
type matrix=array[1..n,1..n]of integer;
var a:matrix;
    i,j:1..n;
```

```
begin
  for i:=1 to n do
  begin
    for j:=1 to n do
      read(a[i,j]);
    readln;
  end;
for i:=1 to n do
begin
  for j:=1 to n do
    write(a[j,i]:5);
  writeln;
end;
end.
```

且运行程序时的输入为:

2□1□3↵

3□3□1↵

1□2□1↵

则程序的输出应是:

2□3□1

1□3□2

3□1□1

例 8、输入 4 名学生数学、物理、英语、化学、pascal 五门课的考试成绩，求出每名学生的平均分，打印出表格。

分析：用二维数组 a 存放所给数据,第一下标表示学生的学号，第二个下标表示该学生某科成绩，如 a[i,1]、a[i,2]、a[i,3]、a[i,4]、a[i,5]分别存放第 i 号学生数学、物理、英语、化学、pascal 五门课的考试成绩,由于要求每个学生的总分和平均分， 所以第二下标可多开两列，分别存放每个学生 5 门成绩和总分、平均分。源程序如下：

```
program ex5_4;
 var a:array[1..4,1..7]of real;  i,j:integer;
 begin
   fillchar(a,sizeof(a),0);      {函数 fillchar 用以将 a 中所有元素置为 0}
   writeln('Enter 4 students score');
   for i:=1 to 4 do
   begin
     for j:=1 to 5 do             {读入每个人 5 科成绩}
     begin
       read(a[i,j]);              {读每科成绩时同时统计总分}
       a[i,6]:=a[i,6]+a[i,j];
     end;
     readln;
     a[i,7]:=a[i,6]/5;{求平均分}
   end;
   writeln( 'No. Mat.  Phy. Eng.  Che. Pas. Tot. Ave.');  {输出成绩表}
   for i:=1 to 4 do
   begin
     write(i:2,' ');
     for j:=1 to 7 do
        write(a[i,j]:9:2);
     writeln;       {换行}
   end;
 end.
```

【上机练习 6.2】

1、输入一个二维数组，找出其中最小的数，输出它的值以及所在行号和列号。
2、输入一 M 行 N 列数组，将第 I 行与第 J 行元素对调（I，J < M）。
3、输入 4×4 方阵，分别求两条对角线上元素之和。
4、矩阵的转置：

```
      A:  ------------------------------------------>  B:
  1   2   3                  转置为          1    4    7    10
  4   5   6                                  2    5    8    11
  7   8   9                                  3    6    9    12
 10   11  12
```

5、给一维数组输入 M 个整数，假设 M=6，数组元素分别为 7 4 8 9 1 5，
要求建立一个如下数组（矩阵）：

```
7  4  8  9  1  5
4  8  9  1  5  7
8  9  1  5  7  4
9  1  5  7  4  8
1  5  7  4  8  9
5  7  4  8  9  1
```

6、建立如下矩阵：

```
2  3  4  5
3  4  5  6
4  5  6  7
5  6  7  8
```

注意：一般情况下，给二维数组赋初值，都是找 A［I，J］与 I 和 J 的函数关系。

```
FOR  I ：= 1   TO   N    DO
    FOR  J：= 1   TO    N    DO
          A［I，J］：=  F（I，J）；
```

本例，对于任意的 A［I，J］= I + J ；所以一定要强调分析能力，让学生学会分析变量的关系。

第三节　数组类型的应用

例 9、输入一串字符，字符个数不超过 100，且以“.”结束。 判断它们是否构成回文。

分析：所谓回文指从左到右和从右到左读一串字符的值是一样的，如 12321，ABCBA，AA 等。先读入要判断的一串字符（放入数组 letter 中），并记住这串字符的长度，然后首尾字符比较，并不断向中间靠拢，就可以判断出是否为回文。

源程序如下：

```
program ex5_5;
var
    letter    :  array[1..100]of char;
    i,j       :  0..100;
    ch        :  char;
begin
  {读入一个字符串以'.'号结束}
  write('Input a string:');
  i:=0;read(ch);
  while ch<>'.' do
  begin
    i:=i+1;letter[i]:=ch;
    read(ch)
  end;
```

```
    {判断它是否是回文}
    j:=1;
    while (j<i)and(letter[j]=letter[i])do
    begin
      i:=i-1;j:=j+1;
    end;
    if j>=i  then writeln('Yes.')
    else writeln('No.');
  end.
```

例 10、奇数阶魔阵

魔阵是用自然数 1，2，3…，n^2填 n 阶方阵的各个元素位置，使方阵的每行的元素之和、每列元素之和及主对角线元素之和均相等。奇数阶魔阵的一个算法是将自然数数列从方阵的中间一行最后一个位置排起，每次总是向右下角排（即 A_{ij} 的下一个是 $A_{i+1,j+1}$）。但若遇以下四种情形，则应修正排数法。

（1）列排完（j=n+1 时），则转排第一列；（2）行排完（即 i=n+1 时），则转排第一行；

（3）对 $A_{n,n}$ 的下一个总是 $A_{n,n-1}$；（4）若 A_{ij} 已排进一个自然数，则排 $A_{i-1,j-2}$。

例如 3 阶方阵，则按上述算法可排成：

4	3	8
9	5	1
2	7	6

有了以上的算法，解题主要思路可用伪代码描述如下：

```
1  i←n div 2+1,y←n  /*排数的初始位置*/
2  a[i,j]←1;
3  for k:=2 to n*n do
4    计算下一个排数位置→(i,j);
5    if a[i,j]<>0 then
6    i←i-1;
7    j←j-2;
6    a[i,j]←k;
7  endfor
```

对于计算下一个排数位置，按上述的四种情形进行，但我们应先处理第三处情况。算法描述如下：

```
1         if (i=n)and(j=n) then
2           j←j-1;   /*下一个位置为(n,n-1)*/;
3         else
4           i←i mod n +1;
5           j←j mod n +1;
6         endif;
```

源程序如下：

```
program ex5_7;
var   a  :array[1..99,1..99]of integer;
      i,j,k,n :integer;
begin
  fillchar(a,sizeof(a),0);
  write( 'n=' );readln(n);
  i:=n div 2+1;j:=n;
  a[i,j]:=1;
  for k:=2 to n*n do
   begin
     if (i=n)and(j=n) then
         j:=j-1
     else
         begin
```

```
                i:=i mod n +1;
                j:=j mod n +1;
              end;
          if a[i,j]<>0 then
              begin
                i:=i-1;
                j:=j-2;
              end;
           a[i,j]:=k;
       end;
    for i:=1 to n do
      begin
        for j:=1 to n do
          write(a[i,j]:5);
        writeln;
      end;
end.
```

【上机练习 6.3】

1. 读入 10 个数，输出偶数项及它们的和，输出奇数项及它们的平均数。
2. 读入 n 个数，打印其中的最大数及其位置号。
3. 有一组（假设有 n 个），其排列顺序如下：3，6，11，45，23，70，67，34，26，89，90，15，56，50，20，10。编一程序交换这数组中任意指定的两段不重合数据。
4. 给定一串整数数列，求出所有的递增和递减子序列的数目，如数列 7，2，6，9，8，3，5，2，1 可分为（7，2），（2，6，9），（9，8，3），（3，5），（5，2，1）5 个子序列，答案就是 5。我们称 2，9，3，5 为转折元素。
5. 设数组 a 是有 n 个元素的整数数组，从中找出最大和子序列。
6. 已知数组 a 中含 n 个整型元素，求 a 中有多少个最大数？多少个次大数？……，多少个 互不相同的数？编程实现之。
7. 试编一个程序，打印出 1000 以内以二进制和十进制正读和反读都一样的整数清单。
8. 约瑟夫问题：n 个人围成一圈，从第一个人开始报数，数到 k 的人出圈。再由下一个人开始报数，数到 k 的人出圈，……依次输出出圈人的编号。N 的值预先设定，k 的值由键盘输入。比如 n=8，k=6，依次出圈的为：6、4、3、5、8、7、2、1。
9. 输入 N 个同学的语、数、英三科成绩，计算他们的总分与平均分，并统计出每个同学的名次，最后以表格的形式输出。
10. 打印杨辉三角形的前 10 行。杨辉三角形形如图：

```
          1                          1
        1   1                      1   1
      1   2   1                    1   2   1
    1   3   3   1                  1   3   3   1
  1   4   6   4   1                1   4   6   6   4   1
```

[图 6-3]　　[图 6-4]

[问题分析]观察图 6-3，大家不容易找到规律，但是如果将它转化为图 6-4，不难发现杨辉三角形其实就是一个二维表的小三角形部分，假设通过二维数组 yh 存储，每行首尾元素为 1，且其中任意一个非首位元素 yh[i,j]的值其实就是 yh[i-1,j-1]与 yh[i-1,j]的和，另外没一行的元素个数刚好等于行数。有了数组元素的值，要打印杨辉三角形，只需要控制一下输出其始位置就行了。

第七章　函数与过程

第一节 函数

PASCAL 给我们提供了一些标准函数，我们不用了解这些函数如何求出来的，只管直接调用它们，挺方便的。如正弦函数，余弦函数，算术平方根..... 有了这些函数，我们觉得很省事。如：求 SIN（1）＋SIN（2）＋... ＋SIN（100）＝？这个程序我们可以这样编写：

例 1、
```
PROGRAM  e1(input,output);
 VAR i:integer;
     s:real;
   BEGIN
      s:=0;
      for i:=1 to 100 do
        s:=s+sin(i);
      writeln('s=',s);
   END.
```

在这个程序里，我们直接用到了正弦函数，至于 SIN（1），SIN（2）如何求出来的我们不需过问，只管直接用它的结果便是了。

我们来看看下面一个例子：求：1！＋2！＋3！＋... ＋10！＝？

如果要编写程序，我们看到求阶乘的操作要执行 10 次，只不过每次所求的数不同。我们想：不至于编写 10 遍求阶乘的程序吧。我们希望有一个求阶乘的函数，假设为 JS（X），那么我们就可以这样求这道题了：

例 2、
```
PROGRAM  e1(input,output);
 VAR i:integer;
     s:real;
   BEGIN
      s:=0;
      for i:=1 to 10 do
           s:=s+js(i);
      writeln('s=',s);
   END.
```

现在的问题是：TURBO　PASCAL 没提供 JS(X)这样一个标准函数，这个程序是通不过的。如果是 PASCAL 的标准函数，我们可以直接调用，如 TRUNC（X），LN（X），SQRT（X）..... 而 PASCAL 提供给我们的可供直接调用的标准函数不多。没关系，我们编写自己的函数！

1.2 函数的编写

自己编写一个函数，它的格式如下：

```
FUNCTION  函数名（形式参数表）：函数类型；
  VAR  函数的变量说明；
  BEGIN
       函数体
  END;
```

我们来分析一下，一个函数的编写可分成三部份：一是函数首部，即第一个语句。它必须以 FUNCTION 开头，函数名是自己取的，取名的原则是便于记忆，和文件名的取名规则类似。形式参数（简称形参）表以标识符的形式给出，相当于函数中的自变量。参数可以有多个，也可以有多种类型。不同类型的参数之间用“；”隔开，同类型的参数如有多个，则用“，”隔开，在其后得加上说明。如：

```
FUNCTION  A1（A，B，C：INTEGER；D，E，F：REAL）：REAL；
```

在最后，函数属于哪种类型，还得表示出来。在本例中，该函数为实型。

第二部分是函数的变量说明部分，对在本函数中将要用到的变量作类型说明，这一点和以前学的变量一样。如果程序不用变量，则此部分也可省掉。

第三部分是函数体，本函数的功能实现就在于此，编写的语句就在里面。

例、编写一求阶乘的函数。

我们给此函数取一名字就叫 JS。

```
FUNCTION js(n:integer):longint;
  var i:integer;
      s:longint;
  begin
    s:=1;
    for i:=1 to n do
      s:=s*i;
   js:=s;
 end;
```

在本例中，函数名叫 JS，只有一个 INTEGER 型的自变量 N，函数 JS 属 LONGINT 型。在本函数中，要用到两个变量 I，S，在 VAR 后已加以说明。在函数体中，是一个求阶乘的语句，但有一点要注意：虽然 N 的阶乘的值在 S 中，但最后必须将此值赋给函数 JS，此时 JS 不带任何参数。在任何函数中，最后都要把最终结果赋给函数名，因为该函数的结果是靠函数名返回的。

在这里，函数的参数 N 是一个接口参数，说得更明确点是入口参数。如果我们调用函数：JS（3），那么在程序里所有有 N 的地方 N 被替代成 3 来计算。在这里，3 就被称为实参。又如：SQRT（4），LN（5），这里 4，5 叫实参。而 SQRT（X），LN（Y）中的 X，Y 叫形参。

1．3 函数的调用

自定义的函数在调用前要先说明，在主程序中的位置如下:

```
PROGRAM  程序名（INPUT，OUTPUT）；
        VAR    主程序变量说明；
        FOUNCTION  函数名（形参表）：函数类型；
          VAR  函数变量说明；
           BEGIN
              函数体
           END；{FUNCTION}
      BEGIN
        主程序
      END .{PROGRM}
```

在主程序中，我们把函数的全部说明放在主程序的变量说明和程序体之间，然后在主程序的执行部分就可以直接调用自定义函数了。注意：在函数的说明部分，我们要用形参，但在程序的执行部分调用自定义函数时，就得用实参了。

例 3 、利用前面定义的阶乘函数，求 5！，9！。

```
PROGRAM e59(input,outout);
  VAR a1,a2:longint;
     fUNCTION js(n:integer):longint;
       var i:integer;
           s:longint;
       begin
         s:=1;
         for i:=1 to n do
           s:=s*i;
         js:=s;
       end;
     BEGIN
        a1:=js(5);
        a2:=js(9);
        writeln('5!=',a1,' ','9!=',a2);
   END.
```

在这个程序中，在主程序的 BEGIN 之前，我们对函数进行了一次说明，在后面的程序中都可以象标准函数那样直接调用自定义函数了。在 FUNCTION 语句中，用的是形参 N，在主程序调用中，调用函数是用的实参，如：JS（5）；程序执行到这儿会自动将 5 代入前面的 FUNCTION 函数中，用 5 取代所有的 N，最

终将结果赋值给 JS。所以在 A1 中一定是 5！，A2 中是 9！。另外，函数不能单独使用，一定要结合主程序才能运行。

如果是求 1！＋2！＋3！＋．．．＋10！，则只需把主程序改成：

```
A1：＝0;
  FOR  J：＝1  TO  10  DO
    A1：＝A1＋JS（J）;
  WRITELN（A1）;
```

在例 3 中，主程序的变量 A1，A2 叫全程变量，它们除了主程序外，还可以在函数中出现；在函数说明中用到的变量 I，S 则是局部变量，只能在函数部分使用，一旦出了函数则失去意义；别外要注意：全程变量和局部变量尽量不要同名。

例 4 、任意输入 10 组三角形的三边，求其面积。

已知三角形的三边，是可以求出面积的。我们可以定义一个已知三角形三边求其面积的函数，设为 AREA（a1,a2,a3）。

```
PROGRAM e5(inmput,output);
  VAR a,b,c,s:real;
      i:integer;
   FUNCTION  area(a1,a2,a3:real):real;
    var s1,d:real;
    begin
      d:=(a1+a2+a3)/2;
      s1:=Sqrt(d*(d-a1)*(d-a2)*(d-a3));
      area:=s1;
    end;
   BEGIN
     for i:=1 to 10 do
       begin
          writeln('input a,b,c');
          readln(a,b,c);
          if  (a+b<=c) or (a+c<=b) or(b+c<=a)
             then writeln('data error!')
               else writeln('s=',area(a,b,c));
       end;
   END.
```

在函数说明中，如果形参的个数不止一个，那么在程序中调用函数的实参个数一定要与形参的个数一致，第一个实参对应第一个形参，第二个实参对应第二个形参．．．次序不能调。

例 5、定义一个函数 CHECK（N，D），让它返回一个布尔值。如果数字 D 在整数 N 的某位中出现则送回 TRUE，否则送回 FALSE。

例如：CHECK（325719，3）＝TRUE;

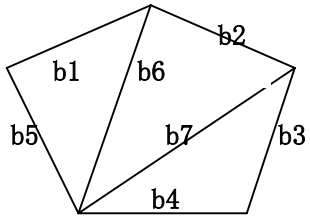
CHECK（77829，1）＝FALSE;

```
PROGRAM e6(input,output);
  VAR a,b:integer;
    FUNCTION check(n,d:integer):boolean;
        var f:boolean;
            e:integer;
         begin
           f:=false;
           while (n>0) and (not f) do
              begin
                e:=n mod 10;
                n:=n div 10;
                if e=d then f:=true;
```

```
            end;
          check:=f;
        end;
BEGIN
   writeln('input n,d');
   read(a,b);
   writeln(check(a,b));
END.
```

例 6、计算如图多边形的面积。

从图中可以看出，五边形的面积是三个三角形面积之和。




```
Program p7_4(input,output);
  Var
      b1,b2,b3,b4,b5,b6,b7,s:real;
  Function area(a,b,c:real):real;
   Var
      P:real;
    Begin
      P:=(a+b+c)/2;
      Area:=sqrt(p*(p-a)*(p-b)*(p-c));
    End;
 BEGIN {主程序}
    Write( 'please input b1,b2,b3,b4,b5,b6,b7:' );
    Readln(b1,b2,b3,b4,b5,b6,b7);
    S:=area(b1,b5,b6)+area(b2,b6,b7)+area(b3,b4,b7);{三次调用函数 area}
    Writeln( 's=' ,s:10:3);
END.
```

函数课堂练习

1. 编程找出由键盘任意输入二个整数中的最大数。
2. 编程找出由键盘任意输入三个整数中的最大数。
3. 求从键盘任意输入两个自然数的最大约数。
4. 求从键盘任意输入三个自然数的最大约数。
5. 求从键盘任意输入两个自然数的最小公倍数。

【上机练习 7.1】

1. 编程求C_K^R = K! / (R!(K-R)!) (K > R > 0)
2. 求正整数2和100之间的完全数。
   完全数：因子之和等于它本身的自然数，如6=1+2+3 ；
3. 哥德巴赫猜想的命题之一是:大于6 的偶数等于两个素数之和。编程将6～100所有偶数表示成两个素数之和。
4. 如果一个自然数是素数,且它的数字位置经过对换后仍为素数,则称为绝对素数,例如 13。试求出所有二位绝对素数

第二节　过程

过程和函数一样，也是子程序。一个过程对应一个需要完成的任务。PASCAL 中提供了不少标准过程，如：READ，WRITE，GET，NEW，PUT. . . . . 这些标准过程在程序中可以直接调用。但仅仅这些标准过程还不能满足我们的需要，我们还要自己定义过程，就象函数一样。但函数必须以值的形式返回，而过程不一定返回一个值，只是执行一个任务而已；函数只能返回一个值，而过程可以返回不止一个值。所以函数不能取代过程。

2．1 过程的定义

过程定义的格式如下：

```
PROCEDURE  过程名（形式参数）；
  VAR  过程的变量说明；
   BEGIN
     过程体
   END;
```

对该格式说明如下：一个过程也分为三部分，1：过程的首部。过程必须以 PROCEDURE 开头，过程名的取名规则和函数名一样，括号里面是形式参数，如形参不止一种，则中间用“；”隔开，同类形参如不止一个，则中间用“，”隔开。另：有时侯过程不用加参数。2：过程的说明部分，用 VAR 开头，它只能对过程中的变量进行说明，同样是局部变量。另：如果过程不用变量，则可将说明部分省略。3：过程体。它是过程的执行部分。

我们来定义一个打印由“＊”组成的矩阵的过程，该矩阵四行五列。

例 7、

```
PROCEDURE print;
    var i,j:integer;
       begin
         for i:=1 to 4 do
           begin
              for j:=1 to 5 do
                  write('*');
              writeln;
            end;
       end;
```

该过程就没有参数，直接执行打印一个固定矩阵的任务，而且也没返回值。

例 8、定义一个求 N! 的过程。

```
PROCEDURE  js(n:integer);
   var s:longint;
       i:integer;
    begin
      s:=1;
      for i:=1 to n do
          s:=s*i;
      writeln(n,'!=',s);
    end.
```

在该过程中，它的值的返回形式和函数不一样：函数是由函数名返回，而过程不是由函数名返回的；在过程的首部不用对过程的类型进行说明。

例 9、定义过程 fa 求 N!。

```
program  exp7_5 ;
  var  x:integer ; t:real ;
  procedure fa(n:integer);
    var
      k:integer;
```

```
    begin
      t:=1;
      for k:=2 to n do
        t:=t*k;
    end;
  begin
    read(x) ;
    fa(x);
    writeln( x:5 , t:8) ;
  end.
```

这里通过全程变量 T，将过程中计算结果传递到 T 变量中。Fa(x)仅仅作为程序中的一条命令被执行。

2．2　过程的调用

自定义过程在程序调用之前要先说明，过程的说明就在主程序的执行语句之前。其格式如下：

```
PROGRAM  程序名（INPUT，OUTPUT）；
  VAR  主程序的变量说明；
    PROCEDURE  过程名（形式参数表）；
      VAR  过程的变量说明；
       BEGIN
        过程体
        END;
    BEGIN
      主程序体
    END.
```

例 10、任意输入 10 个三角形的三边，用过程把其面积求出。

我们把求一个已知三边的三角形的面积定义一个过程，对比一下和例 5 有什么不同。

```
PROGRAM e5(inmput,output);
 VAR a,b,c,s:real;
     i:integer;
    PROCEDURE  area(a1,a2,a3:real);
         var s1,d:real;
         begin
            d:=(a1+a2+a3)/2;
            s1:=Sqrt(d*(d-a1)*(d-a2)*(d-a3));
            writeln('s=',s1);
          end;
      BEGIN
        for i:=1 to 10 do
         begin
           writeln('input a,b,c');
           readln(a,b,c);
           if  (a+b<=c) and (a+c<=b) and (b+c<=a)
               then writeln('data error!')
               else area(a,b,c);
         end;
      END.
```

我们看到：过程的调用和函数不同。函数不能作为独立的一个语句使用，而过程可以。函数的值是由函数名返回的，而过程不能。

现在我们提出一个要求：用过程求出 1！＋2！＋3！＋．．．＋10！＝？

求 N！的问题我们在例 8 已写出来，但阶乘的结果是在过程里用 WRITE 语句输出的，不能用累加语句把结果求出来。那么，怎样用过程将类似的问题求出来？这就得用到变量形参。

2. 3 变量形参

在过程定义的语句中，有个参数表，在参数表中，除了前面我们已用的形参，还有变量形参。变量形参的作用是：它可以作为过程的出口参数。我们可以把过程中求出的结果用变量形参输出到过程外，在过程外面可以调用该参数，因此，该参数是全局变量。其格式上的区别是在变量形参前加上 VAR 即可。

那么我们现在来求 1！＋2！＋3！＋. . . ＋10！＝?

例 11、

```
PROGRAM  e9(input,output);
  VAR  j:integer;s,m:longint;
    PROCEDURE js(n:integer;var m:longint);
      var i:integer;
      begin
        m:=1;
        for i:=1 to n do
          m:=m*i;
      end;
   BEGIN
      s:=0;
      for j:=1 to 10 do
        begin
          js(j,m);
          s:=s+m;
        end;
      writeln('s=',s);
   END.
```

在本例中。我们看到，过程 JS 中用到了变量形参 M，在过程中定义为 LONGINT 类型；而在主程序的变量说明中也得对变量形参 M 说明为同种类型 LONGINGT。于是，在过程中和主程序中都可以用该变量了。

例 12、任意输入一个整数，将它变成字符串输出。如：输入数 34567，打印出字符“34567”。要求用过程的方法实现。

```
PROGRAM E10(input,output);
  var i,k:integer;
      s:string;
    PROCEDURE n_c(n:integer;var s:string);
      var l:inTeger;
      begin
        l:=abs(n);
        s:='';
        repeat
          s:=char((l mod 10) +ord('0'))+s;
          l:=l div 10;
        until l=0;
        if n <0 then s:='-'+s;
      end;
     BEGIN
      for i:=1 to 10 do
        begin
          writeln('input k');
          readln(k);
          n_c(k,s);
          writeln('the string is:',s);
        end;
     END.
```

2.4 形参与变量形参

（1）形参：在函数或过程定义中，没有加 VAR 说明的参数，在调用函数或过程时，调用程序将实参的值直接传递给形参，起着赋值作用。

（2）变量形参：在函数或过程定义中，加有 VAR 说明的参数，在调用函数或过程时，调用程序将实参的变量地址传递给变量形参，因此当过程或函数处理中，改变变量形参的值，则实参的变量值也随之改变。（共享同一个存储单元）

例如下列程序中的参数传递：

```
Program p7_13(input,output);
  var
    x,n:integer;
  procedure chan(x:integer;var y:integer);
    begin
      x:=x+5;
      y:=y+5;
      writeln('x=', x, 'y=', y);{语句②}
   end;
BEGIN   {主程序}
   x:=10;
   n:=10;
   writeln('x=', x, 'n=', n);  {语句①}
   chan(x, n);
   writeln('x=', x, 'n=', n);  {语句③}
END.
```

运行结果：

```
x=10 n=10          {语句①的结果}
x=15 y=15          {语句②的结果}
x=10 n=15          {语句③的结果}
```

过程 chan 中定义了形参 x 和变量形参 y。在调用过程时，形参 x 接受实参的值 10，然后将它加 5，但是形参值的改变并不影响主程序中实参的值（数值传递），所以返回主程序后，输出实参 x 的值仍为 10，可见，实参 x 和形参 x 是两个不同的变量。

y 为变量形参，对于变量形参的操作实际上就是对相应实参 n 的操作（共享存储地址）。y 的初值为 10，调用过程后，值为 15，返回主程序时，值 15 被带回主程序，故 n 也为 15。

例 13、将实数 x 拆分为整数部分 n 和小数部分 p。

```
Program p7_16(input,output);
  var   a1,a2,d1,d2:real;
        z1,z2:integer;
  procedure fen(x: real;var n: integer;var p: real);
    begin
      n:=trunc(x);{n 是整数部分}
      p:=x-n;     {p 是小数部分}
    end;
 begin   {主程序}
    write('input a1,a2: ');
    readln(a1,a2);
    fen(a1,z1,d1);
    writeln(a1:8:3,z1:8,d1:8:3);
    fen(a2,z2,d2);
    writeln(a2:8:3,z2:8,d2:8:3);
 end.
```

运行结果：

```
input a1, a2: 12.87  345.769
   12.87       12     0.870
   345.769     345    0.769
```

小结形参和变量形参的区别：

①形参传值：为形参分配存贮单元，将实参的值赋给形参，过程体内对形参的操作不影响实参的值。一旦过程体执行完毕，系统将收回形参所占用的存贮单元，形参的值也就不复存在。

②变量形参传地址：将实参的地址传给对应的变量形参，即变量形参与实参共享实参的地址，因此对变量形参的操作就是对实参的操作。一旦过程体执行完毕，系统将收回变量形参所占用的存贮单元，但运算结果已保留在对应的实参中。

2.5 变量及其作用域

全程变量：主程序中被说明的变量。

局部变量：在过程或函数中被说明的变量。

在程序中，局部变量、全程变量进行存取的适用范围是不一样的，即作用域不一样。

局部变量的作用域是它们所在的子程序。因形式参数也只在子程序中有效，因此也属于局部变量。

对于局部变量的作用域可以这样理解：当局部变量所在子程序被调用时，局部变量才被分配有效的存储单元；当返回调用程序时，局部变量所占的存储单元就被释放。

全程变量的作用域分为两种情况：

①在全程变量和局部变量不同名时，其作用域是整个程序。

②在全程变量和局部变量同名时，局部变量屏蔽了全程变量。

例 14、变量作用范围：

```
Program p7_10(input,output);
  var
    m:integer;  {m 为全程变量}
  procedure  test2;
   var    m:integer;  {定义 m 为局部变量}
   begin
     m:=100;
   end;
 begin   {主程序}
   m:=5;
   writeln('Before the test2 call,the m is: ',m);{输出过程调用前的 m 值}
   test2;          {调用过程 test2}
   writeln('After the test2 call,the m is: ',m); {输出过程调用后的 m 值}
 end.
```

运行结果：

```
Before the test2 call, the m is:  5
After the test2 call, the m is:  5
```

主程序开始执行后，全程变量 m 被赋初值 5；调用过程 test2 后，由于有与全程变量同名的局部变量 m，故过程中起作用的是局部变量 m，而全程变量 m 暂时被屏蔽不受影响。过程调用完毕返回主程序后，全程变量 m 起作用，局部变量 m 所占用的临时单元被释放。从这个程序可以看出，当过程或函数内包含与全程变量同名的变量时，全程变量的作用域将不包括在此过程或函数中。

2.6 函数和过程的区别

过程和函数都为子程序，但也有区别：

1、标识符不同。函数的标识符为 FUNCTION，过程为：PROCEDURE。

2、函数中一般不用变量形参，用函数名直接返回函数值；而过程如有返回值，则必须用变量形参返回。

3、过程无类型，不能给过程名赋值；函数有类型，最终要将函数值传送给函数名。

4、函数在定义时一定要进行函数的类型说明，过程则不进行过程的类型说明。

5、调用方式不同。函数的调用出现在表达式中，过程调用，由独立的过程调用语句来完成。

6、过程一般会被设计成求若干个运算结果，完成一系列的数据处理，或与计算无关的各种操作；而函数往往只为了求得一个函数值。

四、过程与函数的综合应用

例 15、从键盘读取 A，B 数组的元素，A，B 数组均已从小到大排好序（无相同元素），现将 A，B 合并为数组 C，同样要求数组 C 也是从小到大排好序（有相同元素时只保留一个）。程序中 n 表示数组 A，B 的

长度，i，j，k 分别表示数组 A，B，C 的取数或存数的指针。

```
program excp3;
  const  n=8;   m=2*n;
  type   arr1=array [1..n] of  integer;
         arr2=array [1..m] of  integer;
  var  a, b : arr1;  c : arr2; i, j, k  :integer;
procedure  copy(x: arr1; var  y:arr2;  var  i, j : integer);
    begin
      i:=i+1;
      y[i]:=x[j]; j:=j+1;
   end;
 begin
   for  i:=1  to  n  do  read(a[i] ); readln;
    for  i:=1  to  n  do  read( b[i] ); readln;
   i:=1; j:=1;  k:=0 ;
   while  (I<=n) and (j<=n)  do
     if  a[i]<b[j]  then  copy (a, c, k, i) { 如果 A 数组值小于 B 数组，则将 A 赋值到 C 数
                                                                  组}
              else  if  b[j]<a[i]  then  copy (b, c, k, j) { 将 B 赋值到 C}
                      else begin
                             copy(a, c, k, i);  {A 与 B 的值相同，指针均加 1
                             j:=j+1 ;
                           end;
       while  I<= n  do  copy(a, c, k, i);
       while   j<=n  do  copy(b, c, k, j);
       writeln
       for i:=1 to k do write (c[i]:4);
          writeln;
   end.
```

过程课堂练习

1. 编程找出由键盘任意输入三个整数中的最大数的过程。
2. 输入三个不同的整数，按由小到大排列，用过程编程。
3. 用不带参数的过程和带参数的过程分别编写一个程序，实现变量x和y的值相互交换。
   (数组参数)设计一个过程,将数组中的元素从小到大排列。
4. 编一过程完成二分查找（假设已有一按升序排列的数组）。

【上机练习 7.2】

1. 输入自然数n，求前n个合数（非素数），其素因子仅有2，3，或5。

2. 自然数a的因子是指能整除a的所有自然数，但不含a本身。例如12的因子为：1,2,3,4,6。若自然数a的因子之和为b，而且b的因子之和又等于a，则称a,b为一对“亲和数” 。求最小的一对亲和数。

3. 求前n个自然数的平方和，要求不用乘法。例如：3的平方不用3*3，可用3+3+3。

4. 如果一个数从左边读和从右边读都是同一个数，就称为回文数。例如6886就是一个回文数，求出所有的既是回文数又是素数的三位数。

5. 任何大于2的自然数都可以写成不超过四个平方数之和。如：

$8=2^2+2^2$;　　$14=1^2+2^2+3^2$

由键盘输入自然数 N（2 < N < 2000） ，输出其不超过四个平方数之和的表示式。

第三节　递推算法

递推和递归是编程中常用的基本算法。在前面的解题中已经用到了这两种方法，下面对这两种算法基本应用进行详细研究讨论。

递推算法是一种用若干步可重复的简单运算（规律）来描述复杂问题的方法。

[例 1] 植树节那天，有五位参加了植树活动，他们完成植树的棵数都不相同。问第一位同学植了多少棵时，他指着旁边的第二位同学说比他多植了两棵；追问第二位同学，他又说比第三位同学多植了两棵；…如此，都说比另一位同学多植两棵。最后问到第五位同学时，他说自己植了 10 棵。到底第一位同学植了多少棵树？

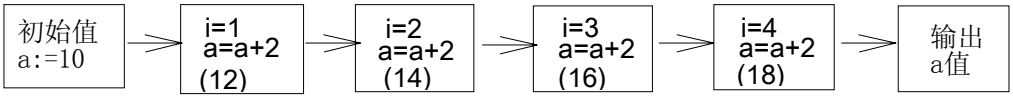
分析：设第一位同学植树的棵数为 a1，欲求 a1，需从第五位同学植树的棵数 a5 入手，根据“多两棵”这个规律，按照一定顺序逐步进行推算：

①a5=10;
②a4=a5+2=12;
③a3=a4+2=14;
④a2=a3+2=16;
⑤a1=a2+2=18;

Pascal 程序：

```
Program Exam1;
  Var i, a: byte;
    begin
      a:=10;                {以第五位同学的棵数为递推的起始值}
      for i :=1 to 4 do     {还有 4 人，递推计算 4 次}
        a:= a+2;            {递推运算规律}
      writeln(' The Num is' , a);
      readln
    end.
```

本程序的递推运算可用如下图示描述：



递推算法以初始{起点}值为基础，用相同的运算规律，逐次重复运算，直至运算结束。这种从“起点”重复相同的方法直至到达一定“边界”，犹如单向运动，用循环可以实现。递推的本质是按规律逐次推出（计算）下一步的结果。

[例 2]斐波列契数列（Faibonacci）0，1，1，2，3，5，8，13，21，34……求此数列第 n 项。

即：F1=0　　(N=1)
　　F2=1　　(N=2)
　　Fn=Fn-1 + Fn-2　　(N>=3)

Pascal 程序：

```
var f0,f1,f2:real;
    i,n:byte;
begin
    readln(n);
    f0:=0;f1:=1;
     for i:=2 to n-1 do
      begin
        f2:=f0+f1;
        f0:=f1; f1:=f2
      end;
     writeln(f2:1:0)
end.
```

【上机练习 7.3】

1、猴子吃枣问题：猴子摘了一堆枣，第一天吃了一半，还嫌不过瘾，又吃了一个；第二天，又吃了剩下的一半零一个；以后每天如此。到第十天，猴子一看只剩下一个了。问最初有多少个枣子？

2、任何一个自然数的立方都可以写成一串连续奇数之和。如：

$1^3=1$

$2^3=3+5=8$

$3^3=7+9+11=27$

$4^3=13+15+17+19=64$

……

编程输入 N，求 N^3 是由哪些奇数之和。

3、楼梯有 N 级台阶，上楼可以一步上一阶，也可以一步上二阶。编一递归程序，计算共有多少种不同走法？

4、兔子在出生两个月以后，就具有生殖后代的能力。假设一对兔子，每月都能生一对兔子，生出来的每一对小兔子，在出生两个月后，也每月生一对兔子。那末，由一对刚出生的小兔子开始，连续不断地繁殖下去，在某个指定的月份有多少对兔子？

5、骨牌铺法：

有 1×n 的一个长方形，用一个 1×1、1×2 和 1×3 的骨牌铺满方格。例如当 n=3 时为 1×3 的方格。此时用 1×1、1×2 和 1×3 的骨牌铺满方格，共有四种铺法。如下图：

ㄣ。

图 4.4.3

递推公式：

边界条件：

第四节　递归算法

一、递归概念

当过程或函数的定义中，其内部操作又直接或间接地出现对自身程序的引用，则称这样的程序嵌套定义为递归定义。

递归算法是把处理问题的方法定义成与原问题处理方法相同的过程，在处理问题的过程中又调用自身定义的函数或过程。

例如，在数学上，所有偶数的集合可递归地定义为：

①0 是一个偶数；

②一个偶数和 2 的和是一个偶数。

可见，仅需两句话就能定义一个由无穷多个元素组成的集合。在程序中，递归是通过函数或过程的调用来实现的。函数或过程直接调用其自身，称为直接递归；函数或过程间接调用其自身，称为间接递归。

二、函数递归

[例 1]仍用上例的计算植树棵数问题来说明递归算法

分析：把原问题求第一位同学在植树棵数 a1，转化为 a1=a2+2;即求 a2；而求 a2 又转化为 a2=a3+2; a3=a4+2; a4=a5+2;逐层转化为求 a2, a3, a4, a5 且都采用与求 a1 相同的方法;最后的 a5 为已知,则用 a5=10 返回到上一层并代入计算出 a4；又用 a4 的值代入上一层去求 a3;...,如此,直到求出 a1。

因此：

$$a_x=\begin{cases}10 & (x=5)\\ a_{x+1}+2 & (x<5)\end{cases}$$

其中求 a_{x+1} 又采用求 a_x 的方法。所以：

①定义一个处理问题的函数 Num(x)：如果 X ＜ 5 就递归调用函数 Num(x+1)；

②当递归调用到达一定条件(X=5)，就直接执行 num:=10，再执行后继语句，遇 End 返回到调用本函数的地方。

③最后返回到开头的原问题，此时所得到的运算结果就是原问题 Num(1)的答案。

Pascal 程序：

```
program exam1_1;
var a:integer;
  function num(x:integer):integer;
    begin
      if x=5 then num:=10
             else num:=num(x+1)+2;
    end;
BEGIN
   writeln('The Num is ',num(1));
   READLN;
END.
```

递归方法说明如下：

①调用原问题的处理过程时，调用程序应给出具体的过程形参值（数据）；

②在处理子问题中，如果又调用原问题的处理过程，但形参值应是不断改变的量(表达式）；

③每递归调用一次自身过程，系统就打开一“层”与自身相同的程序系列；

④由于调用参数不断改变，将使条件满足（达到一定边界），此时就是最后一“层”，不需再调用（打开新层），而是往下执行后继语句，给出边界值，遇到本过程的 END，就返回到上“层”调用此过程的地方并继续往下执行；

⑤整个递归过程可视为由往返双向“运动”组成，先是逐层递进，逐层打开新的“篇章”，（有可能无具体计算值）当最终递进达到边界，执行完本“层”的语句，才由最末一“层”逐次返回到上“层”，每次返回均带回新的计算值，直至回到第一次由主程序调用的地方，完成对原问题的处理。

[例 2] 用递归算法求 X^n 。

分析：把 X^n 分解成：

$X^0 = 1$ (n =0)

$X^1 = X * X^0$ (n =1)

$X^2 = X * X^1$ (n >1)

$X^3 = X * X^2$ (n >1)

…… (n >1)

```
program ex1;
   var x,n:integer;
function cf(n:integer):integer;
begin
   if n=0 then cf:=1
          else cf:=x*cf(n-1);
end;
BEGIN
  write('Please input x,n:');
  readln(x,n);
  write(x,'^',n,' is:',cf(n));
END.
```

递归算法，常常是把解决原问题按顺序逐次调用同一“子程序”(过程)去处理，最后一次调用得到已知数据，执行完该次调用过程的处理，将结果带回，按“先进后出”原则，依次计算返回。

[例 3]用递归函数求 x!

$$x!=\begin{cases}1 & (x=0)\\ x(x-1)! & (x>0)\end{cases}$$

分析：根据数学中的定义把求 x! 定义为求 x*(x-1)!,其中求(x-1)! 仍采用求 x! 的方法，需要定义一个求 x! 的过程或函数，逐级调用此过程或函数，即：

当 x=0 时 x!=1；当 x>0 时，x!=x*(x-1)!。

假设用函数 Fac(x)表示 x 的阶乘，当 x=3 时，Fac(3)的求解方法可表示为：

Fac(3)=3*fac(2)=3*2*Fac(1)=3*2*1*Fac(0)=3*2*1*1=6

①定义递归函数：fac(n: integer): integer;

如果 n=0，则 fac:=1；如果 n>0，则调用函数 fac:=n*fac(n-1);

②返回主程序，打印 fac(x)的结果。

它的执行流程如图 4.4.2 所示

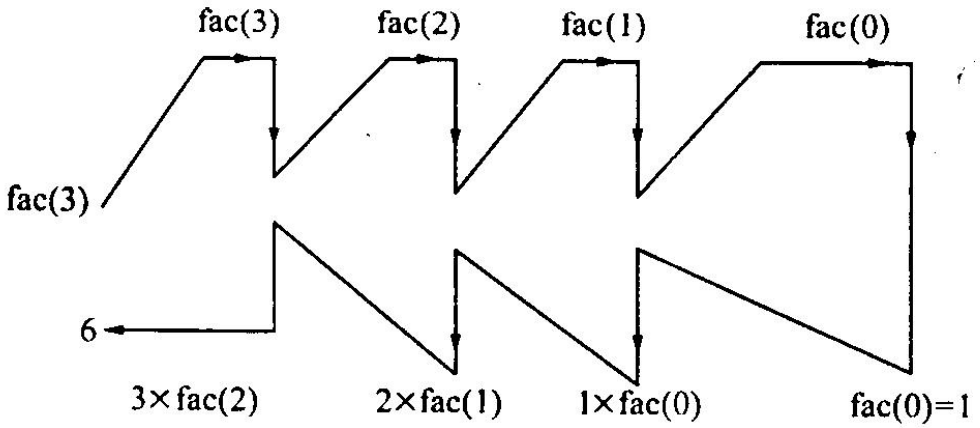


图 4.4.2

```
Program Exam3;
 Var x: integer;
 function fac(n: integer): integer;  {函数 fac(n) 求 n !}
  begin
    if n=0 then fac:=1
      else fac:=n*fac(n-1)   {函数 fac(n-1)递归求(n-1) !}
  end;
 BEGIN
     write(' input x' );  readln(x);
    writeln(x, ' !=' , fac(x));  {主程序调用 fac(x) 求 x !}
 END.
```

递归算法表现在处理问题的强大能力。然而，如同循环一样，递归也会带来无终止调用的可能性，因此，在设计递归过程（函数）时，必须考虑递归调用的终止问题，就是递归调用要受限于某一条件，而且要保证这个条件在一定情况下肯定能得到满足。

[例 4]用递归方法求两个数 m 和 n 的最大公约数。(m>0, n>0)

求两个数的最大公约数，可以用循环搜索方法，找到能被两个数同时整除且是最大的约数的方法；也可以用辗转相除法，这里采用递归程序设计方法：

```
Program  p7_21(input,output);
  var  m,n,g:integer ;
```

```
function  gcd(m,n：integer)：integer;
  var
    r:integer;
  begin
    r:=m mod n;
    if r=0 then gcd :=n
      else gcd:=gcd(n, r);
  end;
BEGIN { 主程序 }
  read (m,n);
  g:= gcd( m,n);
  writeln ( 'm=' ,m ,' n=' ,n ,' gcd=' ,g );
END.
```

运行：

输入 128 48　　输出 m=128 n=48 gcd=16

输入 67 94　　输出 m=67 n=94 gcd=1

三、过程递归

[例 1]仍用上例的计算植树棵数问题来说明递归算法：

分析：把原问题求第一位同学在植树棵数 a1，转化为 a1=a2+2;即求 a2；而求 a2 又转化为 a2=a3+2; a3=a4+2; a4=a5+2;逐层转化为求 a2,a3,a4,a5 且都采用与求 a1 相同的方法;最后的 a5 为已知,则用 a5=10 返回到上一层并代入计算出 a4；又用 a4 的值代入上一层去求 a3;...,如此,直到求出 a1。因此：

$$a_x = \begin{cases} 10 & (x=5) \\ a_{x+1} + 2 & (x<5) \end{cases}$$

其中求 a_{x+1} 又采用求 a_x 的方法。所以:

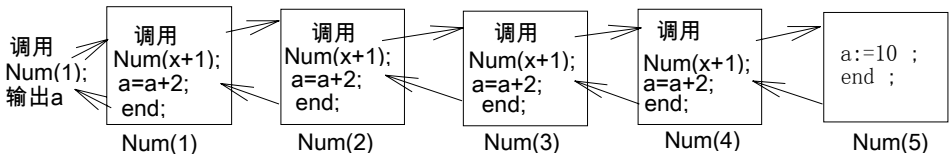
①定义一个处理问题的过程 Num(x)：如果 X ＜ 5 就递归调用过程 Num(x+1);

②当递归调用到达一定条件(X=5)，就直接执行 a :=10，再执行后继语句，遇 End 返回到调用本过程的地方，将带回的计算结果(值)参与此处的后继语句进行运算（a:=a+2）；

③最后返回到开头的原问题，此时所得到的运算结果就是原问题 Num(1)的答案。

```
Program Exam1_1;
  Var a: byte;
  Procedure Num(x: integer);{过程 Num(x)求 x 的棵数}
    begin
      if x=5 then a:=10
        else begin
               Num(x+1);  {递归调用过程 Num(x+1)}
               a:=a+2     {求(x+1)的棵数}
             end
    end;
BEGIN
    Num(1);      {主程序调用 Num(1)求第 1 个人的棵数}
    writeln(' The Num is ' , a);
    readln
END.
```

程序中的递归过程图解如下：



参照图示，递归方法说明如下：

①调用原问题的处理过程时，调用程序应给出具体的过程形参值（数据）；

②在处理子问题中，如果又调用原问题的处理过程，但形参值应是不断改变的量(表达式）；

③每递归调用一次自身过程，系统就打开一“层”与自身相同的程序系列；

④由于调用参数不断改变，将使条件满足（达到一定边界），此时就是最后一“层”，不需再调用（打开新层），而是往下执行后继语句，给出边界值，遇到本过程的 END，就返回到上“层”调用此过程的地方并继续往下执行；

⑤整个递归过程可视为由往返双向“运动”组成，先是逐层递进，逐层打开新的“篇章”，(有可能无具体计算值）当最终递进达到边界，执行完本“层”的语句，才由最末一“层”逐次返回到上“层”，每次返回均带回新的计算值，直至回到第一次由主程序调用的地方，完成对原问题的处理。

[例 2] 用递归算法求 X^n 。

分析：把 X^n 分解成：

$X^0 = 1$ （ n =0 ）

$X^1 = X * X^0$ （ n =1 ）

$X^2 = X * X^1$ （ n >1 ）

$X^3 = X * X^2$ （ n >1 ）

…… （ n >1 ）

$X^n = X * X^{n-1}$ （ n >1 ）

因此将 X^n 转化为：

其中求 X^{n-1} 又用求 X^n 的方法进行求解。

①定义过程 xn(x,n: integer)求 X^n ;如果 n >1 则递归调用 xn (x, n-1) 求 X^{n-1}；

②当递归调用到达 n=0，就执行 t t :=1, 然后执行本“层”的后继语句；

③遇到过程的 END 就结束本次的调用，返回到上一“层”调用语句的地方，并执行其后续语句 tt:=tt*x;

④继续执行步骤③，从调用中逐“层”返回，最后返回到主程序，输出 tt 的值。

Pascal 程序：

```
 Program Exam2;
  Var tt, a, b: integer;
  Procedure xn(x, n: integer); {过程 xn(x, n)求 x^n }
    begin  if n=0 then tt:=1
             else begin
                    xn(x, n-1);   {递归调用过 xn(x,n-1)求 x^(n-1)}
                    tt:=tt*x
                  end;
     end;
  BEGIN
    write(' input x, n:' );  readln(a,b);  {输入 a, b}
    xn(a,b);          {主程序调用过程 xn(a, b)求 a^b}
    writeln(a, ' ^' , b, ' = ', tt);
    readln
  END.
```

递归算法，常常是把解决原问题按顺序逐次调用同一“子程序”(过程)去处理，最后一次调用得到已知数据，执行完该次调用过程的处理，将结果带回，按“先进后出”原则，依次计算返回。

[例 3]用递归过程求 x!

$$x!=\begin{cases} 1 & (x=0) \\ x(x-1)! & (x>0) \end{cases}$$

```
program ex_217;
var
  n:integer;
  t:longint;
```

```
procedure jc(x:longint);
  begin
    if x=1 then t:=1
    else  begin  jc(x-1);t:=t*x;   end;
  end;
  BEGIN
    read(n);
    jc(n);
    writeln(t);
  END.
```

[例 4]用递归算求自然数 A，B 的最大公约数。

分析：求最大公约数的方法有许多种，若用欧几里德发明的辗转相除方法如下：

①定义求 X 除以 Y 的余数的过程;

②如果余数不为 0，则让 X=Y，Y=余数，重复步骤①，即调用过程;

③如果余数为 0，则终止调用过程;

④输出此时的 Y 值。

Pascal 程序：

```
Program Exam4;
  Var a,b,d: integer;
  Procedure Gdd(x, y: nteger);{过程}
    begin
      if x mod y =0 then d :=y
       else Gdd(y, x mod y) {递归调用过程}
    end;

  BEGIN
    write(' input a, b=' ); readln(a, b);
    Gdd(a, b);
    writeln(' (' , a, ' ,' , b, ' )=' , d );
    readln
  END.
```

简单地说，递归算法的本质就是自己调用自己，用调用自己的方法去处理问题，可使解决问题变得简洁明了。按正常情况有几次调用，就有几次返回。但有些程序可以只进行递归处理，不一定要返回时才进行所需要的处理。

（1）递归程序的执行过程

递归程序在执行过程中，一般具有如下模式：

①将调用程序的返回地址、相应的调用前变量保存在栈中；

②执行被调用的程序或函数；

③若满足退出递归的条件，则退出递归，并从栈顶上弹回返回地址、返回变量的值，继续沿着返回地址，向下执行程序；

④否则继续递归调用，只是递归调用的参数发生变化：增加一个量或减少一个量，重复执行直到递归调用结束。

（2）递归程序设计

能够用递归算法解决的问题，一般满足如下要求：

①需要求解的问题可以化为子问题求解，其子问题的求解方法与原问题相同，只是数量的增加或减少；

②递归调用的次数是有限的；必须有递归结束的条件

[深入应用]

例 5、已知一个一维数组 A[1..N]。{N<50} 又已知一整数 M。 如能使数组 A 中任意几个元素之和等于 M，则输出 YES，反之则为 NO。

对于一个已确定的数组 a[1]至 a[n]和一个确定的数 m,判断能否使数组 a 中任意几个元素之和等于 m，等价于判断能否从数组 a 中取任意数使其和为 m。

此时若 a[n]=m，则可以输出“YES”，否则若 n=1，则可以输出“NO”。

否则可以按以下规则进行判断：对于 a 中任意元素 a[n]只有取与不取两种情况：

（1）取 a[n]：

则此时问题转化为：对于一个已确定的数组 a[1]至 a[n-1]和一个确定的数 m-a[n]，判断能否使数组 a 中任意几个元素之和等于 m。

（2）不取 a[n]：

则此时问题转化为：对于一个已确定的数组 a[1]至 a[n-1]和一个确定的数 m，判断能否使数组 a 中任意几个元素之和等于 m。

若用函数 sum(n,m)表示能否从数组 a[1]至 a[n]中取任意数使其和为 m，只要 sum(n-1,m-a[n])和 sum(n-1,m)当中有

一个值为真，则 sum(n,m)为真，否则为假。因此，可以用递归来解此题。

递归终止条件为： if a[n]=m then sum:=true else if n=1 then sum:=false;

```
Program zushu(input,output);
  Const max=50;
  Var
     a:array[1..max] of integer;
     n, m, i:integer ;
  Function sum(n,m:integer):boolean;
    Begin
      if a[n]=m then sum:=true
          else if n=1 then sum:=false
              else sum:=sum(n-1,m-a[n]) or sum(n-1,m);
  End;
  Begin
     write('n=');
     readln(n);
     for i:=1 to n do
        begin
           write('a[',i,']=');
           readln(a[i]);
        end;
     write('m=');readln(m);
     if sum(n,m) then writeln('YES') else writeln('NO');
     readln;
  End.
```

【上机练习 7.4】

1.用递归的方法求 1+2+3+……+N 的值。

2. 用递归函数输出斐波那契数列第 n 项。0,1，1，2，3，5，8，13……

3.输入一个非负整数，输出这个数的倒序数。例如输入 123，输出 321。

4. 用递归算法将数组 A 中的 N 个数倒序输出。

5. 用递归方法求 N 个数中的最大数及其位置。

6. 用递归算法将一个十进制数 X 转换成任意进制数 M（M<=16）。

7. 用递归算法实现二分查找即：有 20 个已经从小到大排序好的数据，从键盘输入一个数 X，用对半查找方法，判断它是否在这 20 个数中。

第八章　集合和记录类型

第一节　集合类型

集合是由具有某些共同特征的元素构成的一个整体。在 pascal 中,一个集合是由具有同一有序类型的一组数据元素所组成，这一有序类型称为该集合的基类型。

一、集合类型的定义和变量的说明

集合类型的一般形式为： set of <基类型>;

说明：①基类型可以是任意顺序类型，而不能是实型或其它构造类型。同时，基类型的数据的序号不得超过 255。例如下列说明是合法的:

```
type letters=set of 'A'..'Z';
numbers=set of 0..9;
s1=set of char;
ss=(sun,mon,tue,wed,thu,fri,sat);
s2=set of ss;
```

②与其它自定义类型一样，可以将类型说明与变量说明合并在一起.如:

```
type numbers=set of 0..9;
var s:numbers;
```

与 `var s:set of 0..9;`等价。

二、集合的值

集合的值是用“[”和“]”括起来，中间为用逗号隔开的若干个集合的元素。如：

[] 空集

[1,2,3]

['a','e','i','o','u']

都是集合。

说明：①集合的值放在一对方括号中，各元素之间用逗号隔开。

②在集合中可以没有任何元素，这样的集合称为空集。

③在集合中，如果元素的值是连续的，则可用子界型的表示方法表示。例如：

[1,2,3,4,5,7,8,9,10,15]

可以表示成： [1..5,7..10,15]

④集合的值与方括号内元素出现的次序无关。例如，[1,5,8] 和 [5,1,8] 的值相等。

⑤在集合中同一元素的重复出现对集合的值没有影响。例如，[1,8,5,1,8] 与 [1,5,8] 的值相等。

⑥每个元素可用基类型所允许的表达式来表示。如 [1,1+2,4]、[ch]、[succ(ch)]。

三、集合的运算

1.赋值运算

只能通过赋值语句给集合变量赋值，不能通过读语句赋值，也不能通过 write(或 writeln)语句直接输出集合变量的值。

2.集合的并、交、差运算

可以对集合进行并、交、差三种运算，每种运算都只能有一个运算符、两个运算对象，所得结果仍为集合。三种运算符分别用“＋”、“＊”、“－”表示。注意它们与算术运算的区别。

3.集合的关系运算

集合可以进行相等或不相等、包含或被包含的关系运算，还能测试一个元素是否在集合中。所用的运算符分别是：＝、<>、>=、<=、 i n

它们都是二目运算，且前 4 个运算符的运算对象都是相容的集合类型，最后一个运算符的右边为集合，左边为与集合基类型相同的表达式。

例 1、设有如下说明：

```
type  weekday=(sun,mon,tue,wed,thu,fri,sat);
      week=set of weekday;
      subnum=set of 1..50;
```

写出下列表达式的值:

(1) [sun,sat] + [sun,tue,fri]
(2) [sun,fri] * [mon,tue]
(3) [wun,sat] * [sun..sat]
(4) [sun] - [mon,tue]
(5) [mon] - [mon,tue]
(6) [sun..sat] - [mon,sun,sat]
(7) [1,2,3,5] = [1,5,3,2]
(8) [1,2,3,4] <> [1..4]
(9) [1,2,3,5] >= [1..3]
(10) [1..5] <= [1..4]
(11) [1,2,3] <= [1..3]
(12) 2 in [1..10]

答: 表达式的值分别是:

(1) [sun,sat,tue,fri]
(2) [ ]
(3) [sat]
(4) [sum]
(5) [ ]
(6) [tue..fri]
(7) TRUE
(8) FALSE
(9) TRUE
(10) FALSE
(11) TRUE
(12) TRUE

例 2、输入一系列字符，对其中的数字字符、字母字符和其它字符分别计数。输入‘？’后结束。

```
program ex10_2;
var id,il,io:integer;    ch:char;
    letter:set of char; digit:set of '0'..'9';
begin
    letter=['a'..'z','A'..'Z'];
    digit:=['0'..'9'];
    id:=0;il:=0;io:=0;
    repeat
      read(ch);
      if ch in letter
      then il:=il+1
      else if ch in digit
           then id:=id+1
           else io:=io+1;
    until ch='?';
    writeln('letter:',il,'digit:',id,'Other:',io);
  end.
```

例 3、调用随机函数产生 10 个互不相同的随机整数（0<=x<=40），放入集合中并一起输出（5 个一行）。

```
program px8_1;
var  a:set of 0..40;
     I,m,n:integer;
   Begin
     A:=[];n:=0;Randomize;
     Repeat
       M:=random(41);
       If not(m in a) then begin a:=a+[m];N:=n+1;End;
```

```
    Until n=10;
    N:=0;
    For I:=0 to 40 do
      If I in a then begin
                       write(I:4);N:=n+1;
                       If (n mod 5) then writeln
                     End;
  End.
```

例 4、编程把一个任意十进制正整数转换成二进制数，要求利用集合表示二进制数。如 18 的二进制数为 10010，则集合的值为[2，5]，即倒数地 2 位和倒数第 5 位是 1，其余位是 0。

输入样例：18=（10010）2

[问题分析] 将十进制正整数 ten 转换成二进制数，只要把 ten 不停地除以 2 求余数，每除 1 次，则 num:=num+1,若求得的余数为 1 则将此时的 num 存入集合中，最后按要求一起输出。

```
program px8_5;
var ten,t,num,I:integer;  two:set of 1..32;
Begin
  write( 'input a integer:' );readln(ten);
  t:=ten; two:=[];num:=0;
  while t<>0 do
  begin
    num:=num+1;
    if t mod 2=1 then two:=two+[num];
    t:=t div 2;
  end;
  if ten=0 then writeln(ten,' =(0)2' )
           else begin
                  write(ten,' =( ');
                  for I:=num downto 1 do
                    if I in teo then write( '1' )
                    else write( '0' );
                    writeln( ')2' );
                end;
  readln;
end.
```

【上机练习 8.1】

1. 调用随机函数产生 10 个互不相同的随机整数（0≤x≤40），放入集合中并一起输出（5 个一行）。
提示：随机函数使用 randomize；{初始化}m:=random(n)；{n，m 都是整数，那么 0≤m≤n-1}
2. 编写一个程序读入一系列字符，将它们分别放在英文字母、数字、其他符号三个集合中，统计出各个集合中元素的个数（区分大小写），并输出这三个集合中的元素。
3. 利用随机函数产生 2 个整数数列 A，B，每个数列包含 20 个不同数（0 到 50 之间），程序要求：（1）找出在 B 中出现而在 A 中没有出现的那些数，并输出；
（2）找出在 B 中出现而在 A 中也出现的那些数，并输出。
4. 输入一个大写字母字符串，找出未在此串中出现的所有大写字母。
5. 编写一个译码程序，将输入的一串字符，（只有小写字母、数字和空格，输入时以句号结束）翻译成原码。译码规则如下：
①数字 0，1，2，3，…，9 分别和字母 a，b，c，…，j 互换；
②字母 k，m，p，t，y 分别和它们的后继互换；
⑶其他字母和空格保持不变。
6. 口袋中有红、黄、蓝、白、黑 5 种颜色的 5 只小球，每次从口袋中取出 3 只球，问：最多有几种不同颜色的组合，并输出每一种方案。

第二节　记录类型

在程序中对于组织和处理大批量的数据来说，数组是一种十分方便而又灵活的工具，但是数组在使用中有一个基本限制，这就是：一个数组中的所有元素都必须具有相同的类型。但在实际问题中可能会遇到另一类数据，它是由性质各不相同的成份组成的，即它的各个成份可能具有不同的类型。例如，有关一个学生的数据包含下列项目：

学号　　字符串类型
姓名　　字符串类型
年龄　　整型
性别　　字符型
成绩　　实型数组

Pascal 给我们提供了一种叫做记录的结构类型。在一个记录中，可以包含不同类型的并且互相相关的一些数据。

（一）记录类型的定义

在 pascal 中，记录由一组称为“域”的分量组成，每个域可以具有不同的类型。

记录类型定义的一般形式：

```
record
   <域名 1>:<类型 1>;
   <域名 2>:<类型 2>;
     :        :
   <域名 n>:<类型 n>;
 end;
```

说明：

①域名也称域变量标识符， 应符合标识符的语法规则。在同一个记录中类型中，各个域不能取相同的名，但在不同的记录类型中，两个类型中的域名可以相同。

②记录类型的定义和记录变量可以合并为一个定义，如：

```
type date=record
            year:1900..1999;
            month:1..12;
            day:1..31
          end;
var x:date;
```

可以合并成:

```
var x: record
         year:1900..1999;
         month:1..12;
         day:1..31
       end;
```

③对记录的操作,除了可以进行整体赋值， 还能对记录的分量——域变量进行。

④域变量的表示方法如下：

记录变量名.域名

如前面定义的记录 X,其 3 个分量分别为:x.year ， x.month ， x.day。

⑤域变量的使用和一般的变量一样， 即域变量是属于什么数据类型，便可以进行那种数据类型所允许的操作。

（二）记录的嵌套

当一个记录类型的某一个域类型也是记录类型的时候，我们说发生了记录的嵌套，看下面的例子：

例 5　某人事登记表可用一个记录表示，其中各项数据具有不同的类型，分别命名一个标识符。而其中的“出生年月日”又包括三项数据，还可以用一个嵌套在内层的记录表示。

具体定义如下:

```
type sexs=(male,female);
     date=record
            year:1900..1999;
            month:1..12;
            day:1..31;
          end;
personal=record
           name:string[15];
           sex:sexs;
           birthdate:date;
           home:string[40];
         end;
```

例 6　设计一个函数比较两个 dates 日期类型记录变量的迟早。

设函数名、形参及函数类型定义为：

AearlyB(A,B:dates):boolean;

函数的形参为两个 dates 类型的值参数。当函数值为 true 时表示日期 A 早于日期 B,否则日期 A 迟于日期 B 或等于日期 B。显然不能对 A、B 两个记录变量直接进行比较，而要依具体的意义逐域处理。

源程序如下：

```
program ex6_7;
type dates=record
             year:1900.1999;
             month:1..12;
             day:1..31
           end;
var x,y:dates;
function AearlyB(A,B:dates):boolean;
var earln:boolean;
begin
  early:=false;
  if (A.year<B.year) then early:=true;
  if (A.year=B.year)and(A.month<B.month)  then early:=true;
  if (A.year=B.year)and(A.month=B.month)and(A.day<B.day)  then early:=true;
  AearlyB:=early;
end;{of AearlyB}
BEGIN
  write('Input DATE X(mm-dd-yy):')readln(X.month,X.day,X.year);
  write('Input DATE Y(mm-dd-yy):')readln(Y.month,Y.day,Y.year);
   if AearlyB(X,Y) then writeln(Date X early!') else writeln('Date X not early!');
END.
```

（三）开域语句

在程序中对记录进行处理时，经常要引用同一记录中不同的域，每次都按 6.4.1 节所述的格式引用，非常乏味。为此 Pascal 提供了一个 with 语句，可以提供引用域的简单形式。

开域语句一般形式：　with <记录变量名表> do
<语句>

功能：在 do 后的语句中使用 with 后的记录的域时，只要直接写出域名即可，即可以省略图 10.2.2 中的记录变量名和“.”。

说明：①一般在 with 后只使用一个记录变量名。如：

```
write('Input year:');  readln(x.year);
write('Input month:'); readln(x.month);
write('Input day:');   readln(x.day);
```

可以改写成:

```
  with x do
    begin
      write('Input year:');readln(year);
      write('Input month:');readln(month);
      write('Input day:');readln(day);
    end;
```

②设 x,y 是相同类型的记录变量，下列语句是非法的：

```
      with x,y do...;
```

③with 后接若干个记录名时,应是嵌套的关系。如有记录说明：

```
   var x:record
          i:integer;
          y:record
               j:0..5;
               k:real;
            end;
          m:real
        end;
```

可以使用:

```
with x do
  begin
    read(i);
    with y do
        read(j,k);
    readln(m);
  end;
```

或简写为:

```
     with x,y do
          readln(i,j,k,m);
```

例 7　读入 10 个日期，再对每个日期输出第二天的日期。输入日期的格式是月、日、年，如 9□30□1993，输出的格式为 10/1/1993。

分析: 可用一个记录变量 today 表示日期。 知道一个日期后要更新为第二天的日期，应判断输入的日期是否为当月的最后一天，或当年的最后一天。

```
  program ex6_8;
     type date=record
                 month:1..12;
                 day:1..31;
                 year:1900..1999;
               end;
      var today:array[1..10]of date;
          i:integer;
          maxdays:28..31;
      begin
        for i:=1 to 10 do {输入 10 个日期}
          with today[i] do
            readln(month,day,year);
        for i:=1 to 10 do
          with today[i] do{求第 i 个日期中月份最后一天 maxdays}
            begin
              case month of
                1,3,5,7,8,10,12:maxdays:=31;
                4,6,9,11 :maxdays:=30;
```

```
            2        :if(year mod 400=0) or( year mod 4=0)and(year mod 100<>0)
                        then maxdays:=29
                        else maxdays:=28;
          end;
          if day=maxdays
            then begin
                   day:=1;
                   if month=12
                     then begin
                            month:=1;year:=year+1;
                          end
                     else month:=month+1;
                 end
            else day:=day+1;
          writeln(month,'/',day,'/',year);
        end;
    end.
```

第三节　综合应用实例

例 8、利用记录类型将 N 个数由大到小排序，输出排序结果并显示每个数原来的位置序号。

```
Program Ex59;
  Type dd=Record                          {定义 DD 记录类型}
           ii:integer;                    {域名 ii 表示数}
           id:integer                     {域名 id 表示数 ii 的位置序号}
         End;
   Var a:array[1..100]of dd;
       i,j,k,n:integer;   t:dd;
   Begin
     Write('Please Input Number of elements:');
     read(n);
     writeln('Input elements:');
     for i:=1 to n do
       begin
         read(a[i].ii);
         a[i].id:=i;
         for j:=1 to i-1 do
           if a[j].ii<a[i].ii then
             begin
               t:=a[i];
               for k:=i-1 downto j do
                 a[k+1]:=a[k];
               a[j]:=t;
             end;
       end;
      for i:=1 to n do
       begin
         write(a[i].ii:5,'(',a[i].id:2,')');
         if i mod 10=0 then writeln
       end;
       readln;writeln;
    End.
```

例 9、编制用筛选法求 1-n(n≤200)以内素数的程序。

分析：由希腊著名数学家埃拉托色尼提出的所谓“筛选法”，步骤如下：

①将所有候选数放入筛中；

②找筛中最小数（必为素数）next，放入集合 primes 中；

③将 next 的所有倍数从筛中筛去；

④重复②～④直到筛空。

编程时，用集合变量 sieve 表示筛子，用集合 primes 存放所有素数。

源程序如下：

```
program ex10_3;
const n=200;
var sieve,primes:set of 2..n;
    next,j:integer;
begin
    sieve:=[2..n];{将所有候选数放入筛中}
    primes:=[];{素数集合置空}
    next:=2;
    repeat
      {找筛 sieve 中最小一个数}
      while not(next in sieve) and(next<=n)do
        next:=succ(next);
      primes:=primes+[next];{将最小数放入素数集合中}
    {将这个素数的倍数从筛中删去}
    j:=next;
    while j<=n do
      begin
        sieve:=sieve-[j];
        j:=j+next;
      end
    until sieve=[];
    j:=0;
    for next:=2 to n do{打印出所有素数}
      if next in primes then
      begin
          write(next:5);
          j:=j+1;
          if j mod 10=0 then writeln;
      end;
    writeln;
end.
```

【上机练习 8.2】

1、输入 20 位学生的数据记录（包含学号、姓名、性别、年龄、成绩五个域），按成绩从高到低排序输出。

第九章 文 件

[内容讲授]

在程序设计中，常常需要从键盘输入大量数据，操作相当麻烦、也很容易出错；同时，在程序运行后也往往会产生大量的输出数据（结果），这给验证结果的正确性和测试程序的对错也带来了很大的麻烦。能不能有一种方法，让程序自动从某个地方读取数据运行，再将程序的运行结果保存到指定的地方呢？当然可以，这就是 Pascal 中的“文件”类型。

在 Pascal 中，文件被定义为同一类型的元素组成的顺序集合。文件所含元素的个数，称为文件的长度。长度为 0 的文件，即不含任何一个元素的文件，称为空文件。

从定义上看，文件与数组很相似，但它们有明显的不同。数组的长度是固定的，而文件的长度不固定，而且可以很长；数组可通过下标对任一数组元素进行访问，而文件一般必须从前往后逐个进行访问，直到找到或找完为止；数组可以整体赋值，而文件却不可以。

文件具有以下 3 个特点：

⑴ 顺序性

文件是由同一类型的元素组成的，它们是按一定的顺序排列和存放的，一般读取文件中的数据或输出文件的内容，都必须从文件头到文件尾顺序访问。在 Pascal 中定义的每个文件类型变量，都自动附带一个文件指针，通过文件指针来指向文件中的某个元素，并对该元素进行操作；

⑵ 永久性

前面讲过的所有数据类型如数组、记录、集合等，它们都是在程序运行时在内存空间临时开辟和存储的，所以不具备永久性；而文件是存储在磁盘等外部存储介质上的，因而它们可以永久地存储起来。并且，数据以文件的形式存放后，还可以被其它程序调用，成为共享数据。

⑶ 容量大

由于内存容量很有限，所以我们在定义和使用数组等数据类型时，数据存储容量往往受到很大的限制；而文件是存储在磁盘等外存中，因而容量可以很大、甚至是无限的。

由于文件有如上特点，文件类型的变量成为一种特殊的变量。文件类型的变量的值可以在程序执行前存在，也可以在程序执行后存在，而且它的值可以比程序本身还大。文件变量是指对文件整体的内容，并且其内容是存放在外存中。对文件的操作只能是对文件中的每个元素进行操作，即由文件指针来对文件的元素进行操作。

一、文件的分类

文件按照不同的原则可以划分成不同的种类。

在 Pascal 中，按文件的结构形式分，文件可以分为文本文件、类型文件和无类型文件。文本文件又称为 text 文件、正文文件，文件的内容是以字符形式（ASCII 码）存放的，因此可以用 DOS 中的 TYPE 命令显示其内容。类型文件又称为 file 文件、二进制文件，它的内容是以二进制代码的形式存放的。无类型文件是一个低层的 I/O 通道，主要用来直接访问固定长度的元素的任意磁盘文件，而不管文件的类型和构造，在信息学奥赛中不会用到这种类型的文件，所以后面也不再介绍了。

如果按照文件的存取方式分，文件又可分为顺序存取文件和随机存取文件两种，即顺序文件和随机文件。顺序文件的的读写都要从文件头开始顺次进行，而随机文件可以从文件的任意指定位置进行读写。在 Pascal 中，文本文件属于顺序存取文件，而类型文件属于随机存取文件。

顺序文件具有如下几个特征：

⑴ 当程序开始把数据输出（写）到文件中去时，总是从文件的起始位置开始。也就是说不可能从文件的中间位置开始存放数据。

⑵ 输出的数据是一个接一个地存放在文件中的。

⑶ 当程序从文件中输入（读取）数据到内存时，也总是从文件的起始位置开始。也就是说不可能从文件的中间位置开始读取数据。

⑷ 读数据时，也必须按存入文件时的顺序一个接一个的读入到内存中。

⑸ 对同一个文件的读写操作不能交叉进行。即不能在读取文件的同时往文件里写，也不能在写入文件的同时从文件里读。

二、文件操作的常用函数和过程

Pascal 提供了一批用于文件操作的标准过程和函数，它们在文件的操作过程中非常有用，下面先列表介绍给大家，后面还将详细介绍它们的使用方法。

1.适合于所有文件类型的标准过程和函数

表 9.1　适合于所有文件类型的标准过程和函数

名　　字	过程或函数	基　　本　　功　　能
assign	过　程	将一个外部文件名赋予文件变量
close	过　程	关闭一个已打开的文件
eof	函　数	返回文件结束状态
erase	过　程	删除一个外部文件
rename	过　程	重新命名一个外部文件名
reset	过　程	打开一个已存在的文件
rewrite	过　程	建立并打开一个新文件

2.只适合于文本文件的标准过程和函数

表 9.2　只适合文本文件的标准过程和函数

名　　字	过程或函数	基　　本　　功　　能
append	过　程	打开一个已存在的文件，从尾部增添元素
eoln	函　数	返回文件的行结束状态
read	过　程	从文本文件中读取一个或多个值赋予相应的变量
readln	过　程	执行 read 过程，然后跳到文件的下一行
write	过　程	将一个或多个值写入文本文件
writeln	过　程	执行 write 过程，然后写入一个行结束标志

3.只适合于类型文件的标准过程和函数

表 9.3　只适合类型文件的标准过程和函数

名　　字	过程或函数	基　　本　　功　　能
read	过　程	从类型文件中读取一个或多个值赋予相应的变量
write	过　程	将一个或多个值写入类型文件中
filepos	函　数	返回指定文件的当前文件位置
filesize	函　数	返回文件的当前长度，即文件中元素的个数
seek	过　程	将文件指针移到指定的位置

三、文件操作的一般步骤

在 Pascal 中，不管使用哪种类型的文件，必须按照以下步骤操作，当然,不同类型的文件操作的具体语句和格式可能略有不同。

1、 在使用文件前，必须对文件类型和变量进行说明；
2、 把磁盘上的实际文件（外部文件）和 Pascal 程序当中的文件（内部文件）建立关联；
3、 打开文件，将文件指针指向开始位置，为文件读写作准备；
4、 对文件进行读、写操作；
5、 在使用完文件后，一定要记住关闭文件，确保文件的完整性和可靠性，否则会引起文件处理错误。切记切记!!!

四、文本文件的操作步骤

1. 文本文件的变量在使用前必须说明其类型为 text；

如：var f1,f2, f3 : text;

2. 使用文本文件前，要先调用 assign 过程，把外部文件名赋予文本文件变量；

assign 过程的调用格式如下：　assign(filevar , filename);

其中，filevar 是文件变量，要在变量说明部分预先定义；filename 是合法的文件名的字符串表达式。assign 过程的作用就是把 filename 赋给文件变量 filevar，在程序中对文件变量 filevar 的操作即为对磁盘文件 filename 的操作。如：

assign(f1, ‘input1.txt’);

注意，input1.txt 是实际的文件名，默认位置在 Pascal 安装目录下，也可以用文件的绝对路径。如：assign(f2 , ‘a:\test1\input1.txt’)；也可以通过 read 语句读入 filename（这儿的 filename 是字符串类型的变量），将文件变量与用户所输入的文件名相联系，如：　read(filename)；assign(f3 , filename);

3. 当将数据写入（输出）到文件时，应先调用 rewrite 或 append 过程打开该文件，再用 write 或 writeln 将实际数据写入到该文件中；

rewrite 过程的调用格式如下： rewrite（filevar）;

它的作用是创建一个新的磁盘文件，并以写的方式打开该文件，初始化文件为空，文件指针指向开始位置。在调用 rewrite 过程前必须先调用 assign 过程，给文件变量一个实在的文件名。

append 过程的调用格式如下： append（filevar）;

append 过程的作用与 rewrite 相似，只是用 append 打开的文件只能向文件尾部添加数据而已。

write 过程的调用格式如下： write（filevar，var1，var2，……，varn）;

它的作用是向文件 filevar 中写入变量 var1，var2，……，varn 的值，当然，文件的元素类型必须和变量类型一致，每向文件中写入一个数据，文件指针向下移一个位置。

writeln 过程的调用格式如下：writeln（filevar，var1，var2，……，varn）;

它的作用是执行 write 过程，然后再写入一个行结束标志。

4. 当需要从文件中读取数据（输入）到内存时，应先调用 reset 过程打开该文件，再用 read 或 readln 将数据读入到内存变量中，且只能从文件的开头读数据；

reset 过程的调用格式如下： reset（filevar）;

它的作用是打开一个已存在的磁盘文件，并将文件指针指向开始位置，表示可以开始读取文件了，但不能向文件写数据，同样 filevar 文件名必须存在，且文件中必须有满足程序运行要求的输入数据。

read 过程的调用格式如下： read（filevar，var1，var2，……，varn）;

它的作用是从文件 filevar 中读取若干个数据，分别赋给变量 var1，var2，……，varn。当然，这里的变量类型必须和文件中元素的类型一致，每读取一个数据，文件指针向下移一个位置。

readln 过程的调用格式如下：readln（filevar，var1，var2，……，varn）;

它的作用是执行 read 过程，然后跳到文件的下一行。

5. 对文件的输入、输出操作必须以行为单位进行，读写完毕要用 close 命令关闭文件。

close 过程的作用是关闭一个文件，无论是向磁盘写文件，还是从磁盘文件中读取数据，读、写完毕后都必须用 close 命令关闭已打开的文件，以保证文件的完整性和可靠性，否则将可能引起文件处理错误。

close 过程的调用格式如下： close(filevar);

注意：

在文本文件的操作过程中，一定要注意并严格区分 read、readln 及 write、writeln 的使用。

五、行结束和文件结束函数

除了上述几个重要的过程外，在文本文件的使用过程中还经常用到两个重要的函数。

Eoln 函数：行结束函数，函数值为布尔型，当文件指针指向回车换行符时，函数值为真（true），否则为假（false）。Eoln 函数一般用在从一个已打开的文件中读取数据时判断一行的数据是否读完。如果文件是以写状态打开的，则 Eoln 函数的值总是假。Eoln 函数的调用形式为：Eoln（filevar）；一般可以省略参数（包括括号）。

Eof 函数：文件结束函数，函数值为布尔型，当文件指针指向文件结束标志（ctrl+z）时，函数值为真（true），否则为假（false）。Eof 函数一般用在从一个已打开的文件中读取数据时判断文件是否结束。Eof 函数的调用形式为：Eof（filevar）；也可省略参数。

六、文本文件的应用举例

[例 9-1] 从键盘输入一段正文，将它复制到指定的磁盘文件中，然后再在显示器上输出。

分析：⑴从键盘输入正文，需逐个输入字符，且按行存储，每行用回车键结束；

⑵当文件写结束时，加一个文件结束标志（ctrl+z），再用 close 关闭文件；

⑶打开此文件，逐个读取字符，并显示在屏幕上。

程序代码如下：

```
program px9_1(input,output);
var   ch:char;
      str1:string[15] ;
      file1:text;
begin
  write( 'please input a file name:' );
```

```
  readln(str1);                                          { 输入文件名、建立新文件 }
  assign(file1,str1);                                    { 将内部变量名与外部文件建立关联 }
  rewrite(file1);                                        { 以写状态打开该文件，准备写入 }
  while not eof  do                                      { 文件未结束（即未从键盘输入 ctrl+z）就写一行 }
     begin
       while not eoln  do                                { 一行未结束（即未从键盘输入一个回车换行符）就继续写 }
         begin
          read(ch);                                      { 从键盘读入一个字符给 ch }
          write(file1,ch);                               { 将 ch 写入文件中 }
         end;
        readln;                                          { 键盘上换一行 }
        writeln(file1);                                  { 写一个行结束符到文件中 }
     end;
  close(file1);                                          { 写文件结束，关闭文件 }
  writeln;                                               { 屏幕换行 }
  reset(file1);                                          { 以读状态再次打开该文件 }
  while not eof (file1)   do                             { 从文件读取数据，直到遇到文件结束符为止 }
     begin
       while not eoln (file1)   do                       { 读取一行数据 }
         begin
          read(file1,ch);                                { 从文件中读一个字符给 ch }
          write(ch: 3);                                  { 将 ch 输出到屏幕上 }
         end;
        readln;                                          { 遇到回车换行符就换一行继续读 }
        writeln;                                         { 屏幕换行 }
     end;
  close(file1);                                          { 读文件结束，关闭文件 }
end.
```

[例 9-2] 将文本文件 input.txt 中的内容复制到一个新的文本文件 output.txt 中。

程序代码如下：

```
program px9_2(input,output);
var f,g:text;
      ch:char;
begin
  assign(f , 'input.txt' );
  assign(g , 'output.txt' );
  reset(f) ; rewrite(g);
  while not eof(f) do
     begin
       while not eoln(f) do
         begin
            read(f,ch);  write(g,ch) ;
         end;
       readln(f); writeln(g);
     end;
  close(f);  close(g);
end.
```

[例 9-3] 设有文本文件 g 中，其中的数据如下(□表示空格)：

```
1□2□3
4□5□6□7
8□9
```

10□11□12
……

试分析下面程序的执行结果。

```
program px9_3(input,output);
var g:text;
    sum1,sum2,i,j,k:integer;
begin
  assign(g , 'input.txt' );
  reset(g);
  sum1:=0;
  for i:= 1 to 3 do
     begin
       while not eoln(g) do
         begin
           read(g,j);
           sum1:=sum1+j
         end;
       readln(g)
     end;
  reset(g);
  sum2:=0;
  for i:= 1 to 3 do
     begin
       for k:=1 to 3 do
         begin
           read(g,j);
           sum2:=sum2+j
         end;
       readln(g)
    end;
  writeln(sum1:6,sum2:6);
  close(g);
end.
```

程序的运行结果为：45　　48

[例 9-4] 产生 n 个随机整数（500 以内），存放在 text 类型文件 file1 中，再从此文件中读取所有数据进行排序，把排好序的数存放在 text 类型文件 file2 中，最后把 file2 中的文件显示器上输出。

分析：程序的流程很清楚，我们模块化的方法来解决，程序代码如下：

```
program px9_4(input,output);
const n=40;
type arr=array[1..40] of integer;
var  s:arr;  x:integer;
     file1,file2:text;

procedure getfile(var f:text) ;                                  { 产生随机数到 f 文件中 }
var i,a:integer;
begin
  assign(f, 'c:\tp\f1.dat') ;                    { 产生的文本文件实际存放在 c:\tp\f1.dat }
  rewrite(f);
  randomize;
  for i:=1 to n do
    begin
      a:=random(501);
```

```
      write(f,a:5);
      if i mod 10 = 0 then writeln(f);
    end;
  close(f);
end;

procedure readfile(var a:arr;var f:text) ;              { 将 f 中的原始数据转存到数组 a 中 }
var i:integer;
begin
  i:=0;
  assign(f, 'c:\tp\f1.dat') ;   reset(f);
  while not eof(f) do
    begin
      i:=i+1;
      read(f,a[i]) ;
      if eoln(f) then readln(f) ;
    end;
  close(f) ;
end;

procedure sort(var a:arr) ;                          { 对数组 a 中的数据利用选择法进行排序 }
var i,j,p,t:integer;
begin
  for i:=1 to n-1 do
    begin
      p:=i;
      for j:=i+1 to n do
        if a[j]<a[p] then p:=j;
      t:=a[i] ;
      a[i]:=a[p] ;
      a[p]:=t;
    end;
end;

procedure writefile(a:arr;var f:text) ;                    { 将排好序的数存放到文件 f 中 }
var i:integer;
begin
  assign(f, 'c:\tp\f2.dat');                { 排好序的的文本文件实际存放在 c:\tp\f2.dat }
  rewrite(f);
  for i:=1 to n do
    begin
      write(f,a[i]:5);
      if i mod 10 =0 then writeln(f);
    end;
  close(f);
end;

procedure writescreen(var f:text);     { 将排好序的文件输出到屏幕上，也可直接输出数组 a }
var a:integer;
begin
  assign(f, 'c:\tp\f2.dat');   reset(f);
  while not eof(f) do
    begin
```

```
      while not eoln(f) do
        begin
          read(f,a);
          write(a:5);
        end;
       readln(f);
       writeln;
    end;
  close(f);
end;

begin {main}
  getfile(file1);
  readfile(s,file1);
  sort(s);
  writefile(s,file2);
  writescreen(file2);
end.
```

随机产生的文件样例：

382	212	452	189	209	206	445	424	25	483
270	69	301	424	341	58	428	470	175	428
174	259	308	455	331	194	345	40	129	367
327	105	140	408	155	330	492	418	500	267

排好序的文件/输出样例：

25	40	58	69	105	129	140	155	174	175
189	194	206	209	212	259	267	270	301	308
327	330	331	341	345	367	382	408	418	424
424	428	428	445	452	455	470	483	492	500

七、类型文件的操作步骤

前面已经讲过，text 文本文件是以 ASCII 码形式存储的，而 file 类型文件是以二进制形式存储的。类型文件与前面讲的文本文件最大的区别就在于类型文件只能存放同一种类型的数据，而文本文件却可以包含多种数据类型的数据项。file 类型文件和 text 文本文件的使用非常相似，file 类型文件的使用大致分为 5 个步骤：

1. file 类型文件的定义；

与 text 类型不同的是，file 是构造类型，file 文件也要先说明文件类型，再说明文件名，格式如下：

```
type    文件类型标识符 = file of 基类型;
var     文件名列表：文件类型标识符;
```

如：

```
type   filetype = file of integer;
var    f1, f2: filetype;
```

也可以将类型说明和变量说明合并在变量说明区内一起加以定义：

```
var     文件名列表 ： file of 基类型;
```

如：var f1, f2 ： file of integer;

其中，file of 为关键字，基类型可以是文件类型以外的其它任何数据类型，它决定了文件中允许存放的数据类型。

2. 用 assign 过程把外部文件名赋予文件变量（同文本文件）；

3. 调用 rewrite 或 reset 过程将文件打开；

这两个过程的使用同 text 文本文件，但区别在于，用 rewrite 或 reset 过程打开 file 文件后，既能向该文件中写数据，同时又能从该文件中读取数据。

4. 用 read 命令进行读操作，用 write 命令进行写操作；

一定要注意，在 file 类型文件中不允许使用 readln 和 writeln 过程，因为在 file 文件中没有行的概念，整个文件就是一行。所以，同样也不能使用 eoln 函数，而只能用 eof 函数。

另外，file 类型文件还提供了两个专用函数，filepos 和 filesize，filepos 函数用来返回指定文件的当前指针位置，格式如下：filepos（文件名）；filesize 函数用来返回指定文件的当前长度，即文件中的元素个数，格式如下：filesize（文件名）；

同时，由于 file 文件是随机文件，可以随机访问文件中的某个元素，所以，Turbo Pascal 为我们提供了一个指针定位过程“seek”，用 seek 过程可以将文件指针先定位到指定的位置，然后再进行读写操作。seek 过程的格式如下：

seek（文件名，位置值/算术表达式）；

如：seek（f1，5）；表示把读写指针定位到 f1 文件的第五个位置。

5. 文件操作完毕，用 close 过程关闭所有打开的文件；

八、类型文件的应用举例

[例 9-5] 设有一个整数文件 f，现要求将其中的偶数乘以 2，奇数减 1，形成一个偶数文件。

分析：有两种方法，一是另外生成一个偶数文件 g，一是将生成的偶数文件仍然保存在原文件 f 中，下面分别给出这两个程序代码：

```
program px9_5_1(input,output);                          { 生成一个偶数文件 g }
var  f,g : file of integer;
     x:integer;
begin
  assign(f,' input.txt' );
  assign(g,' output.txt' );
  reset(f);   rewrite(g);
  while not eof(f)  do
     begin
       read(f,x);
       if odd(x) then x:=x-1
                 else x:=x+x;
       write(g,x);
     end;
  close(f) ; close(g);
end.

program px9_5_2(input,output);                  { 将生成的偶数文件仍然保存在原文件 f 中 }
var  f : file of integer;
     x, i:integer;
begin
  assign(f,' input.txt' );
  reset(f);
  i:=1;                                                 { 人为设定一个写指针 }
  while not eof(f)  do
     begin
       Read(f,x);                                       { 系统指针作为读指针 }
       If odd(x) then x:=x-1
                 else x:=x+x;
       seek(f,i);                                   { 将写指针定位到目标位置 }
      write(f,x);
      i:=i+1;
    end;
  close(f);
end.
```

[例 9-6] 设有两个已经排好序（从小到大）的整数文件 t1 和 t2，请编程将 t1 和 t2 合并成一个性文件 t，使得合并后的文件也是有序的（从小到大）。

程序代码如下：

```
program px9_6(input,output);
type intfile = file of integer;
var  t1,t2,t:intfile;

procedure paixu(var f,g,h:intfile);
  var m,x,y:integer;
  begin
    reset(f) ; reset(g) ; rewrite(h) ;
    m:=0;                                                    { m为初始标志 }
    while (not eof(f) and not eof(g)) do
      begin
        if m=0 then begin  read(f,x) ; read(g,y) ; m:=1;    end;
        if x<y then begin  write(h,x); read(f,x);   end
               else begin  write(h,y); read(g,y);   end;
      end;
    while not eof(f) do begin  read(f,x); write(h,x);   end;
    while not eof(g) do begin  read(g,y); write(h,y);   end;
    close(f); close(g); close(h);
  end;

begin  { main }
  assign(t1,' input1.txt' );
  assign(t2,' input2.txt' );
  assign(t, ' output.txt' );
  paixu(t1,t2,t);
end.
```

注意：类型文件用写字本或 TYPE 命令打开看，全是乱码（二进制代码）。

【上机练习 9.1】

1. 写一个程序统计文本文件 f 中的行数和字符数。
2. 写一个程序，统计文本文件中的单词个数以及以小写字母 a，b 开头的单词个数。
3. 设有一个文本文件中存放着一组整数（≤100 个），现将该组数读入到 x 数组中，再将它按从大到小的顺序排列，然后再将排序了的数存入到原文件中。
4. 从键盘输入 n 个学生的记录（记录项有学号、语文、数学、英语、总分），存入到 file 类型文件中，求出每个学生的总分，最后将所有数据输出到屏幕上。
5. 设有一个实数文件 f，求文件中的所有数之和。
6. 写一个程序，判断两个整数文件是否相同。
7. 随机产生 20 个三位数，存入一个整数类型文件中，然后将文件中的数读出，如果该数是素数则将其乘以 2 再存入原文件中，否则将数据减 5 再存入原文件中。
8. 调用随机函数，分别产生 20 个 1 到 200 之间的随机整数存入到 file1 和 file2 中，再将上述两个文件中的 40 个数存入到 file3 中，要求 file3 中的数有序排列。

第十章　字符串处理

第一节　字符与字符串类型

一、字符、字符串类型的使用

（一）字符类型

字符类型为由一个字符组成的字符常量或字符变量 。

字符常量定义:

```
const
    字符常量=‘字符’
```

字符变量定义:Ⅱ

```
Var
    字符变量:char;
```

字符类型是一个有序类型，字符的大小顺序按其 ASCⅡ代码的大小而定。函数 succ、pred、ord 适用于字符类型。

例如：后继函数：succ(‘a’)= ‘b’
　　前继函数：pred（‘B’）= ‘A’
　　序号函数：ord（‘A’）=65

例 1、按字母表顺序和逆序每隔一个字母打印。即打印出：

```
a  c  e  g  I  k  m  o  q  s  u  w  y
z  x  r  v  t  p  n   l   j  h  f  d  b
```

程序如下：

```
program ex8_1;
  var letter:char;
  begin
    for letter:='a' to 'z' do
    if (ord(letter)-ord('a'))mod 2=0 then write(letter:3);
    writeln;
    for letter:='z' downto 'a' do
    if (ord(letter)-ord('z'))mod 2 =0 then write(letter:3);
   writeln;
  end.
```

分析：程序中，我们利用了字符类型是顺序类型这一特性，直接将字符类型变量作为循环变量，使程序处理起来比较直观。

（二）字符串类型

字符串是由字符组成的有穷序列。

字符串类型定义：

```
type  <字符串类型标识符>=string[n];
var
      字符串变量: 字符串类型标识符;
```

其中:n 是定义的字符串长度,必须是 0~255 之间的自然整数,第 0 号单元中存放串的实际长度,程序运行时由系统自动提供,第 1~n 号单元中存放串的字符。若将 string[n]写成 string，则默认 n 值为 255。

例如：

```
type
    man=string[8];
    line=string;
var
     name：man;
    screenline：line;
```

另一种字符类型的定义方式为把类型说明的变量定义合并在一起。

例如：VAR
```
        name：STRING[8]；
        screenline：STRING；
```
Turbo Pascal 中，一个字符串中的字符可以通过其对应的下标灵活使用。
例如：
```
var
  name：string；
begin
  readln (nsme)；
  for  i：=1 to ord (name[0]) do
    writeln (name[i])；
end.
```
语句 writeln (name[i]) 输出 name 串中第 i 个字符。

例 2、求输入英文句子单词的平均长度。
```
program ex8_2;
  var  ch:string;
       s,count,j:integer;
  begin
    write('The sentence is :');
    readln(ch);
    s:=0;
    count:=0;
    j:=0;
    repeat
      inc(j);
      if not (ch[j] in [':',',',';','''','!','?','.',' ']) then inc(s);
      if ch[j] in[' ','.','!','?'] then inc(count);
    until (j=ord (ch[0])) or (ch[j] in ['.','!','?']);
    if ch[j]<>'.' then writeln('It is not a sentence.')
    else writeln('Average length is ',s/count:10:4);
  end.
```
分析:程序中，变量 s 用于存句子中英文字母的总数，变量 count 用于存放句子中单词的个数，ch[j] 表示 ch 串中的第 j 个位置上的字符，ord（ch[0]）为 ch 串的串长度。程序充分利用 Turbo Pascal 允许直接通过字符串下标得到串中的字符这一特点，使程序比较简捷。

第二节 字符串的操作

一、字符串的运算和比较

由字符串的常量、变量和运算符组成的表达式称为字符串表达式。
字符串运算符包括：
1. +：连接运算符
例如：‘Turbo ’ + ‘PASCAL’ 的结果是 ‘Turbo PASCAL’。
若连接的结果字符串长度超过 255，则被截成 255 个字符。若连接后的字符串存放在定义的字符串变量中，当其长度超过定义的字符串长度时，超过部份字符串被截断。
例如：var
```
      str1，str2，str3：string[8]；
    begin
      str1：= ‘Turbo ’；
      str2：= ‘PASCAL’；
      str3：=str1+str2；
    end.
```
则 str3 的值为：‘Turbo PA’。

2. =、〈〉、〈、〈=、〉、〉=：关系运算符

两个字符串的比较规则为，从左到右按照ASCⅡ码值逐个比较，遇到ASCⅡ码不等时，规定ASCⅡ码值大的字符所在的字符串为大。

例如：‘AB’〈‘AC’ 结果为真；

‘12’〈‘2’ 结果为真；

‘PASCAL ’ = ‘PASCAL’ 结果为假；

例3、对给定的10个国家名，按其字母的顺序输出。

```
program ex8_3;
  var i,j,k:integer;
      t:string[20];
      cname:array[1..10] of string[20];
  begin
    for i:=1 to 10 do readln(cname[i]);
    for i:=1 to 9 do
      begin
        k:=i;
        for j:=i+1 to 10 do
          if cname[k]>cname[j] then k:=j;
        t:=cname[i];cname[i]:=cname[k];cname[k]:=t;
      end;
    for i:=1 to 10 do writeln(cname[i]);
  end.
```

分析：程序中，当执行到 if cname[k]>cname[j]时，自动将 cname[k]串与 cname[j]串中的每一个字符逐个比较，直至遇到不等而决定其大小。这种比较方式是计算机中字符串比较的一般方式。

二、字符串的函数和过程

Turbo Pascal 提供了八个标准函数和标准过程，见下表，利用这些标准函数与标准过程，一些涉及到字符串的问题可以灵活解决。

函数和过程名	功 能	说 明
copy(s,m,n)	取 s 中第 m 个字符开始的 n 个字符	若 m 大于 s 的长度，则返回空串；否则，若 m+n 大于 s 的长度，则截断
length(s)	求 s 的动态的长度	返回值为整数
pos(sub,s)	在 s 中找子串 sub	返回值为 sub 在 s 中的位置，为 byte 型
insert(sour,s,m)	在 s 的第 m 个字符位置处插入子串 sour	若返回串超过 255，则截断
delete(s,m,n)	删除 s 中第 m 个字符开始的 n 个字符串	若 m 大于 s 的长度，则不删除；否则，若 m+n 大于 s 的长度，则删除到结尾
str(x[:w[:d]],s)	将整数或实数 x 转换成字符串 s	w 和 d 是整型表达式，意义同带字宽的 write 语句
val(s,x,code)	将字符串 S 转换成整数或实数 x	若 S 中有非法字符，则 code 存放非法字符在 S 中的下标；否则，code 为零。code 为整型
upcase(ch)	将字母 ch 转换成大写字母	若 ch 不为小写字母，则不转换

例4 效对输入日期(以标准英语日期,月/日/年)的正确性，若输入正确则以年．月．日的方式输出。

```
program  ex8_4;
 const max:array[1..12] of byte
       =(31,29,31,30,31,30,31,31,30,31,30,31);
 var  st:string;
      p,w,y,m,d:integer;
 procedure err;
   begin
```

```
      write('Input Error!');
      readln;
      halt;
    end;
  procedure init(var x:integer);
    begin
      p:=pos('/',st);
      if (p=0) or (p=1) or (p>3) then err;
      val(copy(st,1,p-1),x,w);
      if w<>0 then err;
      delete(st,1,p);
    end;
  begin
    write('The Date is :'); readln(st);
    init(m);
    init(d);
    val(st,y,w);
    if not (length(st)<>4) or (w<>0) or (m>12) or (d>max[m]) then err;
    if (m=2) and (d=29)
      then if y mod 100=0
            then begin
                   if y mod 400<>0 then err;
                 end
      else if y mod 4<>0 then err;
    write('Date :  ',y,'.',m,'.',d);
    readln;
  end.
```

分析：此题的题意很简单，但在程序处理时还需考虑以下几方面的问题。

1. 判定输入的月和日应是 1 位或 2 位的数字，程序中用了一个过程 inst，利用串函数 pos，求得分隔符/所在的位置而判定输入的月和日是否为 1 位或 2 位，利用标准过程 val 判定输入的月和日是否为数字；

2. 判定月和日是否规定的日期范围及输入的年是否正确；

3. 若输入的月是 2 月份，则还需考虑闰年的情况。

例 5　对输入的一句子实现查找且置换的功能。

```
program  ex8_5;
var s1,s,o:string;
    i:integer;
begin
  write('The text:'); readln(s1);
  write('Find:');readln(s);
  write('Replace:');readln(o);
  i:=pos(s,s1);
  while i<>0 do begin
    delete(s1,i,length(s));
    insert(o,s1,i);
    i:=pos(s,s1);
  end;
  writeln(s1);
  readln;
end.
```

分析:程序中,输入要查找的字符串及要置换的字符串,充分用上了字符串处理的标准过程 delete、insert 及标准函数 pos。

第三节 字符串的综合应用

例 6、判断从键盘输入的字符串是否为回文（从左到右和从右到左读一串字符的值）是一样的，如 ABCDCBA，1234321，11，1），串长< 100 ，且以点号 ‘. ’ 结束。

2000 年竞赛题：判断一个数是否为回文数。

```
VAR  LETTER: ARRAY [ 1.. 100 ]  OF   CHAR ;
      I, J: 0.. 100 ;
      CH: CHAR ;
BEGIN
   WRITELN (‘INPUT  A  STRING :  ’);
     I: = 0 ; READ (CH);
     WHILE    CH  < >  ‘. ’   DO
       BEGIN
          I: =I+1 ;
         LETTER[ I ] : =  CH ;
         READ (CH);
       END;
     J: =1 ; { I 指向数组的尾部，J 指向数组的头部 ，逐个比较 }
     WHILE  (J < I)  AND  (LETTER[ J ]= LETTER[ I ])  DO
        BEGIN
          I: = I  -  1 ;
         J: =J + 1
       END;
     IF   J > = I   THEN   WRITELN (‘YES’)
                   ELSE   WRITELN (‘NO ’);
END.
```

例 7：统计一个英文句子中有多少英文单词。假设句子中字符数小于 80 个，单词间用至少一个空格隔开（可以有多个空格），句子头部可以有多个空格，句子以 ‘*’ 作为结束标志。

如：‘□□□THIS□□IS□A□□GOOD□□□BOOK! * ’

```
CONST  N=80 ;
VAR STR: STRING[ N ];
     PREC: CHAR;  NUM, IINTEGER;
BEGIN
   READLN (STR);
   PREC: =BLANK;    NUM: =0;
   FOR  I: =1   TO   N   DO
      BEGIN
        IF    (STR[ I ] < > BLANK) AND (PREC=BLANK)  THEN  NUM: =NUM+1;
        PREC: =STR[ I ]
      END;
   WRITELN (‘NUM  OF   WORDS   IS:  ’, NUM: 3 );
END.
```

例 8：设姓名最多有 15 个字符组成，其中姓氏至多 5 个字符，姓与名之间用空格分隔，输入若干姓名，以‘*’结束，统计各种姓氏的个数。

分析：为统计所读姓名中不同姓氏的人数，首先需要 把读入的姓氏保存在一张表 TABLE 中，读入一个姓名时，先要到 TABLE 中查一下是否已有相同姓氏了，如果有只需将人数加 1；如果没有则 TABLE 的长度加 1，将此姓添加到表中，同时将此姓的人数置为 1，重复以上过程，直到读到‘*’结束。

```
CONST   N=20;     {  N  要足够大}
        B5= '□□□□□';
VAR  I, SIZE, P: INTEGER; FIND: BOOLEAN;
         TABLE: ARRAY[ 1.. N]   OF  ARRAY [1.. 5 ]  OF  CHAR ;
          NUM: ARRAY[ 1 .. N]    OF  INTEGER;
         NAME: ARRAY[1 .. 15]  OF  CHAR;
          XING: ARRAY[1 .. 5]  OF  CHAR;
BEGIN
   FOR  I: = 1  TO   N   DO  TABLE [ I ]: = B5 ;
   SIZE: =0;
   WRITELN ('INPUT  NAMES:   '); READLN (NAME);
   WHILE  NAME [ 1 ] < >  '*'  DO
   BEGIN
      XING: =B5; P: =1;
       WHILE (P < = 5) AND  (NAME [ P ] < > '□') DO
          BEGIN
            XING [ P ]: =NAME [ P ];
            P: =P+1
          END;
       FIND: =FALSE;   I: =0 ;
      WHILE   (NOT  FIND) AND (I < SIZE)   DO
          BEGIN
            I: =I+1;
            IF  TABLE [ I ] = XING  THEN  FIND: =TRUE
          END;
      IF   FIND  THEN  NUM [ I ]: =NUM [ I ]+1
                ELSE  BEGIN
                        SIZE: =SIZE+1;
                        TABLE [ SIZE ]: =XING ;
                        NUM [ SIZE ]: = 1
                      END;
      READLN (NAME);
   END;
   FOR  I: = 1   TO   SIZE   DO
      IF  I MOD 2 = 0  THEN  WRITELN (B5, TABLE [ I ] , NUM [ I ]: 3)
                       ELSE   WRITE (TABLE [ I ], NUM [ I ] : 3);
   WRITELN
END.
```

第十一章　算法初步

第一节　回溯算法

一、 回溯法的思想

在求解一些问题（如走迷宫、地图着色等问题）时，题目的要求可能是求出原问题的一种或所有可能的解决方案。这类问题的解往往是由一个一个的步骤或状态所构成的，每一步骤又有若干种可能的决策方案；由于没有固定、明确的数学解析方法，往往要采用搜索的做法，即从某一个初始状态出发，不断地向前（即下一个状态）搜索，以期最终达到目标状态，从而得到原问题的一个解或所有的解。在搜索的过程中，由于问题本身及所采取的搜索方法的特点（如在缺乏全局及足够的前瞻信息的情况下进行搜索等），会导致走到某一状态就走不下去的情况，这时，就必须回头（即回到上一步，而不是回到最初的状态），再尝试其他的可能性，换一个方向或方法再试试。这样，不断地向前探索、回溯，再向前、再回溯，直至最终得出问题的解，或者一路回溯到出发点（出现这种情况即表示原问题无解）。注意，这种搜索过程并不是尝试搜索问题解空间中所有的可能状态和路径，而是采用深度优先的方式，沿着一条路径，尽可能深入地向前探索。

二、 实现要点

由于回溯法是一种搜索方法，而且是一种深度优先式的搜索算法，在搜索过程中，前进和回退交织进行，因而，必须有适当的手段和机制来已经走过的状态，以便在发生回溯时能够回退到上一步，再选择与上一步不同的方向继续探索下去。要满足回溯过程中对状态进行记录的要求，最好的数据结构便是堆栈了，它与回溯时只回退到上一步非常吻合。（在具体实现时，栈又可以采用数组、链表等结构来实现。）此外，还要有适当的数据结构来表示整个问题的可能解空间，以及在每一步（每一状态）时可能的选择。

在理解和把握回溯法时，首先要能够手工地理解和模拟在搜索过程中，实际的前进和后退的过程，真正以人工的方式想通实际发生的前进-回退过程。然后，在编制程序时，就象递归算法一样，从较为宏观而自然的层面上，来表达出这种不断进-退的过程，即可构造出有效的回溯算法。

在回溯算法中，把握好何时发生回溯（即发生回溯的条件）是非常关键的，应注意有效而正确地表达回溯条件。还要注意，原问题的状态空间、每一步可能的行动方案、记录已走过的所有步骤的数据结构等，都是非常关键的。

此外，从本质上看，回溯算法符合递归算法，问题的解决可以转化为子问题，其子问题的解法与原问题相同，只是数据增大或减少；因此，在实现回溯算法时，又常常可以考虑采用递归算法来实现，这时，算法中所涉及的堆栈等数据结构就由系统来维护了。

三、应用举例

回溯法可以用来解决一系列问题，如自然数的排列、n 皇后问题、迷宫问题、数的拆分、0/1 背包问题、旅行商问题、货船装货问题、图形覆盖正方形等。下面我们就来看两个应用实例。

【例 1】马的遍历

中国象棋半张棋盘如图 1（a）所示。马自左下角往右上角跳。今规定只许往右跳，不许往左跳。比如图 4（a）中所示为一种跳行路线，并将所经路线打印出来。打印格式为：

0，0->2, 1->3, 3->1, 4->3, 5->2, 7->4, 8…

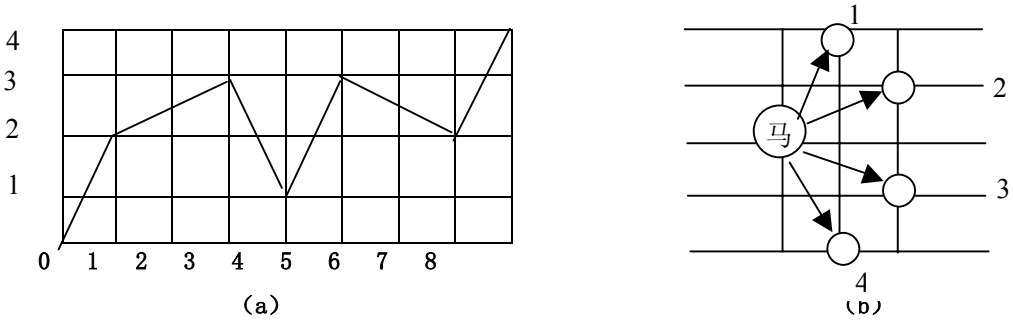


图 1

【算法分析】

如图 4（b）,马最多有四个方向，若原来的横坐标为 j、纵坐标为 i,则四个方向的移动可表示为：

1: (i,j) → (i+2,j+1); (i<3,j<8)

2: (i,j) → (i+1,j+2); (i<4,j<7)

3: (i,j) → (i-1,j+2); (i>0,j<7)

4: (i,j) → (i-2,j+1); (i>1,j<8)

搜索策略：

S1:A[1]:=(0,0);

S2:从A[1]出发，按移动规则依次选定某个方向，如果达到的是（4,8）则转向 S3,否则继续搜索下一个到达的顶点；

S3:打印路径。

【算法设计】

```
procedure try(i:integer);                {搜索}
var  j:integer;
begin
  for j:=1 to 4 do                       {试遍 4 个方向}
    if  新坐标满足条件 then
       begin
         记录新坐标;
         if  到达目的地 then  print      {统计方案，输出结果}
            else try(i+1);               {试探下一步}
         退回到上一个坐标，即回溯;
       end;
end;
```

【参考程序】

```
program exam2;
const  x:array[1..4,1..2] of integer=((2,1),(1,2),(-1,2),(-2,1)); {四种移动规则}
var  t:integer;                          {路径总数}
     a:array[1..9,1..2] of integer;      {路径}
procedure print(ii:integer);             {打印}
var   i:integer;
begin
  inc(t);                                {路径总数}
  for i:=1 to ii-1 do
     write(a[i,1],',',a[i,2],'-->');
  writeln('4,8',t:5);
  readln;
end;
procedure try(i:integer);                {搜索}
var  j:integer;
begin
  for j:=1 to 4 do
    if (a[i-1,1]+x[j,1]>=0) and (a[i-1,1]+x[j,1]<=4) and
       (a[i-1,2]+x[j,2]>=0) and (a[i-1,2]+x[j,2]<=8) then
       begin
         a[i,1]:=a[i-1,1]+x[j,1];
         a[i,2]:=a[i-1,2]+x[j,2];
         if (a[i,1]=4) and (a[i,2]=8) then  print(i)
            else try(i+1);               {搜索下一步}
         a[i,1]:=0;a[i,2]:=0
       end;
end;
```

```
BEGIN                                    {主程序}
  try(2);
END.
```

【例 2】选书

学校放寒假时，信息学竞赛辅导老师有 A，B，C，D，E 五本书，要分给参加培训的张、王、刘、孙、李五位同学，每人只能选一本书。老师事先让每个人将自己喜欢的书填写在如下的表格中。然后根据他们填写的表来分配书本，希望设计一个程序帮助老师求出所有可能的分配方案，使每个学生都满意。

学生	A	B	C	D	E
张同学			Y	Y	
王同学	Y	Y			Y
刘同学		Y	Y		
孙同学				Y	
李同学		Y			Y

【算法分析】

可用穷举法，先不考虑“每人都满意” 这一条件，这样只剩“每人选一本且只能选一本”这一条件。在这个条件下，可行解就是五本书的所有全排列，一共有 5! =120 种。然后在 120 种可行解中一一删去不符合“每人都满意”的解，留下的就是本题的解答。

为了编程方便，设 1，2，3，4，5 分别表示这五本书。这五个数的一种全排列就是五本书的一种分发。例如 54321 就表示第 5 本书（即 E）分给张，第 4 本书（即 D）分给王，……，第 1 本书（即 A）分给李。“喜爱书表”可以用二维数组来表示，1 表示喜爱，0 表示不喜爱。

【算法设计】

S1：产生 5 个数字的一个全排列；

S2：检查是否符合“喜爱书表”的条件，如果符合就打印出来；

S3：检查是否所有的排列都产生了，如果没有产生完，则返回 S1；

S4：结束。

上述算法有可以改进的地方。比如产生了一个全排列 12345，从表中可以看出，选第一本书即给张同学的书，1 是不可能的，因为张只喜欢第 3、4 本书。这就是说，1×××× 一类的分法都不符合条件。由此想到，如果选定第一本书后，就立即检查一下是否符合条件，发现 1 是不符合的，后面的四个数字就不必选了，这样就减少了运算量。换句话说，第一个数字只在 3、4 中选择，这样就可以减少 3/5 的运算量。同理，选定了第一个数字后，也不应该把其他 4 个数字一次选定，而是选择了第二个数字后，就立即检查是否符合条件。例如，第一个数选 3，第二个数选 4 后，立即检查，发现不符合条件，就应另选第二个数。这样就把 34×××一类的分法在产生前就删去了。又减少了一部分运算量。

综上所述，改进后的算法应该是：在产生排列时，每增加一个数，就检查该数是否符合条件，不符合，就立刻换一个，符合条件后，再产生下一个数。因为从第 I 本书到第 I+1 本书的寻找过程是相同的，所以可以用递归方法。算法设计如下：

```
procedure try(i);
begin
   for j:=1 to 5 do
     begin
        if 第 i 个同学分给第 j 本书符合条件 then
          begin
             记录第 i 个数
             if i =5 then 打印一个解
                    else try(i+1);
                删去第 i 个数字
           end;
        end;
end;
```

【参考程序】

```
program exam3;
type five=1..5;
const
     like:array[five,five] of 0..1=((0,0,1,1,0),(1,1,0,0,1),
                                    (0,1,1,0,0),(0,0,0,1,0),(0,1,0,0,1));
     name:array[five] of string[6]=('zhang','wang','liu','sun','li');
var
     book:array[1..5] of 0..5;
     flag:set of five;
        c:integer;

procedure print;
var i:integer;
begin
  inc(c);writeln('answer',c,':');
  for i:=1 to 5 do
      writeln(name[i]:10,':',chr(64+book[i]));
end;

procedure try(i:integer);
var j:integer;
begin
  for j:=1 to 5 do
     if not(j in flag) and (like[i,j]>0) then
     begin
        flag:=flag+[j];book[i]:=j;
        if i=5 then print
               else try(i+1);
        flag:=flag-[j];book[i]:=0;
     end;
end;
begin
  flag:=[];c:=0;try(1);
  readln
end.
```

输出结果：

```
zhang: C
wang: A
liu: B
sun: D
li: E
```

第二节 贪心算法

一、贪心法的思想

在实际问题中，经常会遇到求一个问题的最优解，这就是所谓的最优化问题。最优化问题往往包含一组限制条件和一个优化函数，符合条件的解决方案称为可行解，使优化函数取得最佳值的可行解称为最优解。

贪心法是求解这类问题的一种常用算法，它的思想和做法是这样：从问题的某一个初始解出发，采用逐步构造（迄今为止）最优解的方法向给定的目标前进。在每个局部阶段，都做出一个看上去最优的决策（即某种意义下的、或某个标准下的局部最优解），并期望通过每次所做的局部最优选择，能够产生出一个全局最优解来。做出贪心决策的依据称为贪心准则（策略）。

贪心与递推不同的是，推进的每一步不是依据某一固定的递推公式，而是做一个当时看似最佳的贪心选择，从而不断地将问题实例归纳为更小的相似子问题。

在有些最优化问题中，采用贪心法求解不能保证一定得到最优解，这时可以采取一些变形的贪心法或其他解决最优化问题的方法(如动态规划方法)。

下面我们通过几个应用实例来看看贪心法的应用。

二、应用举例

【例 3】删数问题

通过键盘输入一个高精度的正整数 n(n 的有效位数≤240)，去掉其中任意 s 个数字后，剩下的数字按原左右次序将组成一个新的正整数。编程对给定的 n 和 s，寻找一种方案，使得剩下的数字组成的新数最小。

输入：n

s

输出：最后剩下的最小数

【样例输入】

178543

S=4

【样例输出】

13

【问题分析】

由于正整数 n 的有效位数最大可达 240 位，所以可以采用字符串类型来存储 n。那么，应如何来确定该删除哪 s 位呢？是不是只要删掉最大的 s 个数字就可以了呢？

为了尽可能地逼近目标，我们选取的贪心策略为：每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字，否则删除第一个递减区间的首字符。然后回到串首，按上述规则再删除下一个数字。

重复以上过程 s 次，剩下的数字串便是问题的解了。例如：对 n=178543，s=4，删数的过程如下：

n=178543　　{删掉 8}

n=17543　　{删掉 7}

n=1543　　{删掉 5}

n=143　　{删掉 4}

n=13　　{解为 13}

这样，删数问题就与如何寻找递减区间首字符这样一个简单的问题对应起来了。

要注意一个细节问题：可能会出现字符串首部有若干个 0(甚至整个字符串都是 0)的情况。按以上贪

心策略编制的程序框架如下：

```
输入 s, n;
while s > 0 do
  begin
    i:=1;  {从串首开始找}
    while (i < length(n)) and (n[i]<n[i+1]) do  i:=i+1;
    delete(n,i,1);  {删除字符串 n 的第 i 个字符}
    s := s - 1;
  end;
while (length(n)>1) and (n[1]= '0' ) do delete(n,1,1);   {删去串首可能产生的无用零}
输出 n;
```

【例 4】0/1 背包问题

有一个容量为 weight 的背包，现在要从 n 件物品中选取若干件装入背包中，每件物品 i 的重量为 w[i]，价值为 p[i]。定义一种可行的背包装载为：背包中物品的总重不能超过背包的容量，并且一件物品要么全部选取，要么不选取。定义最佳装载是指所装入的物品价值最高，并且是可行的背包装载。

【样例输入】

```
11                    {weight}
4                     {n}
2   4   6   7         {w[i]}
6  10  12  13         {p[i]}
```

【样例输出】

```
0  1  0  1
23
```

【问题分析】

设有数组 chosen[1..n]，若 chosen[i]=1，表示物品 i 被装入了背包中，chosen[i]=0 表示物品 i 不装入背包中。那么，chosen[0,1,0,1]就是一种可行的背包装载方案，也是一种最佳的装载方案，此时的总价值为 23。

【算法分析】

0/1 背包问题有好几种贪心策略，每种贪心策略都是采用多步过程来完成背包的装入，在每一步中，都是利用某种固定的贪心准则来选择将某一件物品装入背包。

一种贪心准则为：从剩余的物品中，选出可以装入背包的价值最大的物品。这种贪心准则不能保证得到最优解。例如，weight=105，n=3，w=[100,10,10]，p=[20,15,15]，按照以上这种“价值贪心准则”，获得的解为 choice=[1,0,0]，这种方案的总价值为 20，而最优解为 choice=[0,1,1]，其总价值为 30。

另一种方案是“重量贪心准则”，即从剩下的物品中，选择可以装入背包的重量最小的物品。虽然这种规则对于前面的例子能产生最优解，但在一般情况下，不一定能得到最优解，例如，weight=25，n=2，w=[10,20]，p=[5,100]。获得的解为 choice=[1,0]，总价值为 5，比最优解 choice=[0,1]的总价值 100 要差。

本题还有一种方案，即“单位价值贪心准则”，这种方案是从剩余物品中，选择可装入包的 p[i]/w[i]值最大的物品，这种策略也不能保证得到最优解。例如，weight=30，n=3，w=[20,15,15]，p=[40,25,25]。获得的解为 choice=[1,0,0]，总价值为 40，而最优解为 choice=[0,1,1]的总价值为 50。

其实，0/1 背包问题是一个复杂的 NP 问题。对于这类问题，也许根本就不存在多项式时间的算法。在用贪心法解这类问题时，我们还可以结合其他的一些优化策略，如启发式策略等，使解的结果与最优解相差在一定的范围内，也就是，可以得到原问题的近似最优解。另外，在解本题这样的问题时，还可以采用动态规划方法。

第三节 分治算法

一、分治法的思想与要点

为了说明分治法的思想，我们先来看一个问题：

如果有 16 枚硬币，其中有一枚是伪造的，并且那枚伪造硬币的重量和真硬币的重量不同(假设比真币轻)。你能不能用最少的比较次数找出这枚伪造的硬币?为了帮助你完成这一任务，将提供一台可用来比较两组硬币重量的仪器，利用这台仪器，可以知道两组硬币的重量是否相同。

要解决这一问题，可以两两比较。或者，也可以这样来找：将全部硬币分成两组，一次比较两组。一次比较后，可以舍弃完全是真币的那一组，只对另一组进行下一步的比较。在这种做法下，问题的规模明显缩小了，而且每一次比较的规模都成倍地减少。在这种方法下，参与比较的硬币越多，处理起来就越快，投机性也大大减少。

解决方法的关键在于能将大问题分割成若干小问题，小问题与原有问题是完全类似的。通常，我们将这种大化小的策略称为分治法（“分而治之”）。分治法在设计查找、排序等算法时很有效，最常用的分治法有二分法、归并法、快速排序等。

二、应用举例

【例 5】 循环比赛

设有 N 个选手进行循环比赛，其中 $N=2^M$，要求每名选手要与其他 N-1 名选手都赛一次，每名选手每天比赛一次，循环赛共进行 N-1 天，要求每天没有选手轮空。

输入：M

输出：表格形式的比赛安排表

【样例输入】3

【样例输出】

```
1 2 3 4 5 6 7 8
2 1 4 3 6 5 8 7
3 4 1 2 7 8 5 6
4 3 2 1 8 7 6 5
5 6 7 8 1 2 3 4
6 5 8 7 2 1 4 3
7 8 5 6 3 4 1 2
8 7 6 5 4 3 2 1
```

【问题分析】

以 M=3(即 $N=2^3=8$)为例，可以根据问题要求，制定出如下图所示的一种方案：

	第一天	第二天	第三天	第四天	第五天	第六天	第七天
1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	A 1	2	7	8	B 5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	C 5	6	3	4	D 1	2
8	7	6	5	4	3	2	1

以表格的中心为拆分点，将表格分成 A、B、C、D 四个部分，就很容易看出有 A=D，B=C，并且，这一规律同样适用于各个更小的部分。

这样，对 8 个队的赛事排列就逐步减化为对 4 个队的排列，然后又进一步减化为对 2 个队的排列，最后就成为 1 个队的排列。因此，这一问题可以采用分治法的思想来解决。

在设计程序时，采用由小到大的方法进行扩展，而数组下标的处理是解决该问题的关键。

【算法过程】

```
Begin
    N:=2^m;
    对数组 A 的第一行赋值;
    for s:=1 to k do   {s 表示层数}
    begin
        n:=n/2;
        for t:=1 to n do
           for i:=m+1 to 2*m do
             for j:=m+1 to 2*m do
             begin
                a[i,j+(t-1)*m*2]:= a[i-m,j+(t-1)*m*2-m];
                a[i,j+(t-1)*m*2-m]:= a[i-m,j+(t-1)*m*2];
             end;
         m:=m * 2;
      end;
  end.
```

【例 6】快速排序

（1）基本思想

在当前无序区 R[1..H]中任取一个数据元素作为比较的"基准"(不妨记为 X)，用此基准将当前无序区划分为左右两个较小的无序区：R[1..I-1]和 R[I+1..H]，且左边的无序子区中数据元素均小于等于基准元素，右边的无序子区中数据元素均大于等于基准元素，而基准 X 则位于最终排序的位置上，即 R[1..I-1]≤X.Key≤R[I+1..H](1≤I≤H)，当 R[1..I-1]和 R[I+1..H]均非空时，分别对它们进行上述的划分过程，直至所有无序子区中的数据元素均已排序为止。

（2）排序过程【示例】

初 始 关键字　　[49 38 65 97 76 13 27 49]
第一次交换后　　[27 38 65 97 76 13 49 49]
第二次交换后　　[27 38 49 97 76 13 65 49]
J 向左扫描，位置不
变，第三次交换后 [27 38 13 97 76 49 65 49]
I 向右扫描，位置不
变，第四次交换后 [27 38 13 49 76 97 65 49]
J 向左扫描　　[27 38 13 49 76 97 65 49]
（一次划分过程）
初 始 关键字　[49 38 65 97 76 13 27 49]
一趟排序之后　[27 38 13] 49 [76 97 65 49]
二趟排序之后　[13] 27 [38] 49 [49 65] 76 [97]
三趟排序之后　13 27 38 49 49 [65] 76 97
最后的排序结果 13 27 38 49 49 65 76 97
各趟排序之后的状态

快速排序算法

```
Const     n=10;
Var    a:array [1..n] of integer;  k:integer;
procedure qsort(low,high:integer);
 var  i,j:integer;x:integer;
 begin
   if low<high then
     begin
       i:=low;j:=high;x:=a[i];
       repeat
         while (a[j]>=x)and(i<j)do j:=j-1;                    {把大于基准的数留在右面}
         if (a[j]<x)and(i<j) then
           begin
             a[i]:=a[j];i:=i+1;                               {把小于基准的数交换到左面}
           end;
         while (a[i]<=x)and (i<j) do i:=i+1;                  {把小于基准的数留在左面}
         if (a[i]>x)and(i<j) then
           begin
             a[j]:=a[i];j:=j-1;                               {把大于基准的数交换到右面}
           end;
       until i=j;                                             {直到 I 与 j 重叠，完成一次分组过程}
       a[i]:=x;                                               {把基准数插入相应的位置，以后不再参与排序}
       qsort(low,i-1);                                        {小于基准的数重复分组}
       qsort(i+1,high);                                       {大于基准的数重复分组}
     end;
   end;
```

算法改进：

```
procedure qsort(l,r:integer);
  var i,j,mid:integer;
  begin
    i:=l;j:=r; mid:=a[(l+r) div 2];                           {将当前序列在中间位置的数定义为中间数}
    repeat
      while a[i]<mid do inc(i);                               {在左半部分寻找比中间数大的数}
      while a[j]>mid do dec(j);                               {在右半部分寻找比中间数小的数}
      if i<=j then begin                                      {若找到一组与排序目标不一致的数对则交换它们}
                     swap(a[i],a[j]);
                     inc(i);dec(j);                                                {继续找}
                   end;
    until i>j;
    if l<j then qsort(l,j);                                   {若未到两个数的边界，则递归搜索左右区间}
    if i<r then qsort(i,r);
  end;{sort}
```

从输出的数据可以发现，对较坏的初始数据如第一组输入数据，快速排序在交换数据的次数方面已给出了相当好的结果。时间的复杂性是 $O(n\log^2 n)$，速度快，但它也是不稳定的排序方法。

第四节 穷举算法

一、穷举法的思想与要点

穷举法也称为“枚举法”，这种算法基本思想是依题目的部分条件，确定答案的大致范围；在此范围内，对所有可能的情况一一列举，逐一验证，直到全部情况验证完毕，或者得到了需要的结果。

若某个情况经验证符合题目的全部条件，则它就是本题的一个答案；若全部情况经验证后，都不符合题目的全部条件，则原题无解。

用穷举法解题的大致步骤如下：

1）分析题目，确定所要求的解是什么？

2）确定解的可能取值范围是什么？

3）穷举出所有可能的解

4）验证每一个可能的解

5）优化

二、应用举例

下面我们来看一个非常简单、但又能较好地说明穷举法的应用的例子。

【例 7】九头鸟(传说中的一种怪鸟，它有九个头、两只脚)、鸡和兔子关在一个笼子里。数数它们的头正好是 100，数数它们的脚数也正好是 100。请编程计算其中九头鸟、鸡和兔子各有多少只？

【问题及方法分析】

基本题意与已知条件，设九头鸟有 M 只，鸡有 N 只，兔子有 L 只，则有：

$$9M+N+L=100 \quad (1)$$

$$2M+2N+4L=100 \quad (2)$$

题目的要求就是要找出符合条件的 M、N 和 L。该如何入手来解决这个问题呢？可以采用穷举法来解决本题。

【方法 1】:

由于 M、N 和 L 的变化范围都是 1~100，因此，可以采用穷举法的思想与做法，让 M、N 和 L 分别从 1 变化到 100（利用多重循环来实现），看看它们的每一组取值是否能满足上面两个等式的要求。如果满足要求，则将 M、N、L 的值打印出来；否则，继续搜索。

```
begin
  t：= 0            {记录共有多少组满足条件的数据}
  writeln(‘九头鸟’, ‘鸡’, ‘兔’);
  for m:=1 to 100 do          {穷举各种可能的取值}
    for n:= 1 to 100 do
      for l := 1 to 100 do
        if (9*m+n+l=100) and (2*m+2*n+4*l=100) then
          begin
            writeln(m,n,l);
            t:=t+1
          end;
  writeln(‘总共有：’,t,’种情况.’ );
end;
```

【方法 2】:

仍然采用穷举法，只是先做一些预先的分析，从而可以达到一定的优化效果。由题意可知：

如果 100 个头都是九头鸟的话，那九头鸟最多有 100 div 9=11 只；如果 100 只脚都是鸡的话，则最多有 50 只鸡。同理，兔子最多有 25 只。

```
t = 0
for m:=1 to 11 do
  for n: = 1 to 50 do
     for l:= 1 to 25 do
        if (9*m+n+l=100) and (2*m+2*n+4*l=100) then
          begin
            writeln(m, n, l);
            t:=t+1
          end;
  writeln(‘总共有：’, t, ‘种情况.’);
end.
```

在上面的方法 2 里，采用的是缩短循环变量的初值和终值之间距离的方法，来减少循环的次数，以减少程序的运行时间。

那么，能不能对方法 2 做进一步的优化呢?

【方法 3】:

由方法 2 可知，鸡的循环变量范围最大，最值得进一步优化以缩小其取值范围。

根据等式(1)，将 N 用 M 和 L 来表示，得到：

$$N=100-9M-L \qquad (3)$$

这样一来,就又可以做进一步的优化了:将原来的三重循环降低到了两重循环(减少了一个循环变量)。

但是，(3)式根据 M 和 L 计算出来的 N 值可能会出现负数，这是不允许的。所以，在循环体中判断(2)式是否成立时，还要加上 N>0 这一条件。

```
t:= 0
for m:=1 to 11 do
  for l:= 1 to 25 do
    begin
      n:=100 - 9*m - l
      if (n>0) and (2*m+2*n+4*l=100) then begin writeln(m,n,l); t:=t+1; end;
    end;
writeln(t);
```

【方法 4】: 进一步的优化

将式(1)×2 再减去(2)，这样就消去 N ，得到：

$$16M - 2L = 100$$

整理上式可得：

$$L = 8M - 50 \qquad (4)$$

这样就可以将两重循环进一步简化成一重循环。

但是，(4)式由 M 计算出来的 L 也可能会出现负数，这是不允许的，故在循环体中的语句是：

```
if (l>0) and (n>0) then begin writeln(m,n,l); t:=t+1; end;
```

表达式(4)应写在表达式(3)的前面，即必须先求出 l，才能求 n。

```
t:= 0;
for m:=1 to 11 do
  begin
    l:= 8*m  - 50;
    n:=100-9*m-l;
    if (l>0) and (n>0) then begin writeln(m,n,l); t:=t+1; end;
  end;
writeln(t);
```

上面的四种方法都能给出正确的结果，但所需循环次数分别是 100*100*100、11*50*25、11*25、11 次，通过一系列优化措施，以及将不必要的循环去掉，使得程序的效率不断地提高。

提高穷举法效率的方法

1） 在步长值不变的情况下，减少循环变量的初值和终值的差距；

2） 在初值、终值不变的情况下，增大步长值；

3） 减少循环嵌套的层数。

具体使用时，应根据题目的条件，灵活运用。

【例 8】装箱问题（NOIP 2001）

【问题描述】

有一个箱子的容量为 V（V 为正整数，且满足 0≤V≤20000），同时有 n 件物品（0<n≤30），每件物品的体积值为正整数。

要求从 n 件物品中，选取若干装入箱内，使箱子的剩余空间最小。

输入：1 行整数，第 1 个数表示箱子的容量，第 2 个数表示有 n 件物品，后面 n 个数分别表示这 n 件物品各自的体积。

输出：1 个整数，表示箱子剩余空间。

【输入输出样例】

输入：

```
24  6  8  3  12  7  9  7
```

输出：

```
0
```

【问题分析】

此问题有数种解法，此处我们来看看如何用穷举法解法这个问题。

由于将不同的物品装入箱子有多种组合方案，每一种组合方案的结果（即箱子有剩余空间）都不完全相同，因此，对这样的问题，可以考虑采用穷举的方法，一一列举出各种可能的组合，来分别计算出其结果，最后选择最优的组合。

按照上面给出的用穷举法解题的套路或模式，我们可以分析出要求的解是各件物品的某种选取组合，使得箱子的剩余空间最小；这一解的取值范围即是各件物品都有可能被选中或不被选中。因此，可以通过列举各件物品的被选取或不被选取的状态可能，来尝试每一种可能的解。就本题而言，解的验证相对较为容易，只要列举出某一种可能的解（即物品组合）后，计算一下在该解下的箱子剩余空间大小即可（因为本题的要求不仅仅是找出一种可行解，更是找出一种最优解）。

【数据结构】

```
w: array[1..30] of integer;   {存储 n 件物品的体积}
a: array[0..30] of integer;
                  {存储 n 件物品的选取状态，1 表示选取，0 表示未选取，a[0]作为循环控制开关}
min: integer;     {记录最小剩余量}
x: integer;
```

【算法描述】

对数组 a 中各元素的穷举过程如下表所示：

状态	a[0]	a[1]	a[2]	…	a[n-1]	a[n]
初始状态	0	0	0	…	0	1
2	0	0	0	…	1	0
3	0	0	0	…	1	1
…	…	…	…	…	…	…
…	0	1	1	1	1	1
2^n	1	0	0	0	0	0

对每一种状态，计算箱子所用容量（$s=\sum_{i=1}^{n}(a[i]\times w[i])$），如果 v －s ＜ min，则更正 min 的值。

最后，输出最终的 min 即可。

【参考程序】

```
program p7;
var v: integer;
    n: integer;
    w: array[1..30] of integer;
    a: array[0..30] of integer;
    i, j, min, s: integer;
    f: text;
begin
  assign(f, 'input.txt' );
  reset(f);
  read(f,v);
  min:=v;  {初始时，设定空间最小剩余量为箱子总容量}
  read(f, n);
  for i:=1 to n do read(f, w[i]);
  close(f);
  for i:=0 to n do a[i]:=0;
  while a[0]=0 do
  begin
    j:=n;
    while a[j]=1 do j:=j-1;      {逢 1 进位}
    a[j]:=1;                     {设定新的状态}
    for i:=j+1 to n do a[i]:=0;
    s:=0;                        {计算物品的体积和}
    for i:=1 to n do s:=s+a[i]*w[i];
    if (v-s>0) and (v-s<min) then min:=v - s;  {min 存储了目前最少的剩余量}
  end;
  writeln(min);
end.
```

上面给出的是本题的简单穷举解法，在该解法的基础上，可以进行一些优化，如将物品按体积排序，一旦发现体积总和超出了容量要求，则可以不再向下列举。

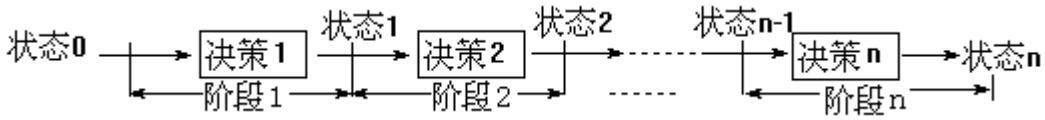
其实，对本题而言，穷举搜索并不是最好的方法。由于本题给出了 V 的规模，因此，可以采用动态规划方法。

第五节　动态规划

动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法。不象前面所述的那些搜索或数值计算那样，具有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。因此读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。我们也可以通过对若干有代表性的问题的动态规划算法进行分析、讨论，逐渐学会并掌握这一设计方法。

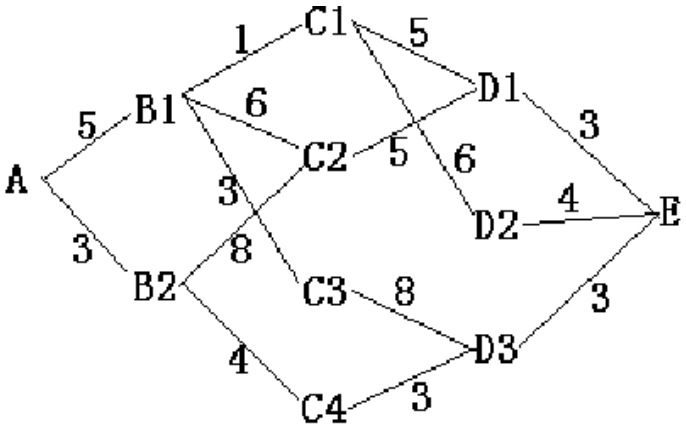
一、多阶段决策过程的最优化问题

在现实生活中，有一类活动的过程，由于它的特殊性，可将过程分成若干个互相联系的阶段，在它的每一阶段都需要作出决策，从而使整个过程达到最好的活动效果。当然，各个阶段决策的选取不是任何确定的，它依赖于当前面临的状态，又影响以后的发展，当各个阶段决策确定后，就组成一个决策序列，因而也就确定了整个过程的一条活动路线，这种把一个问题看作是一个前后关联具有链状结构的多阶段过程就称为多阶段决策过程，这种问题就称为多阶段决策问题。如下图所示：



多阶段决策过程，是指这样的一类特殊的活动过程，问题可以按时间顺序分解成若干相互联系的阶段，在每一个阶段都要做出决策，全部过程的决策是一个决策序列。要使整个活动的总体效果达到最优的问题，称为多阶段决策问题。

【例 9】最短路径问题。下图给出了一个地图，地图中的每个顶点代表一个城市，两个城市间的一条连线代表道路，连线上的数值代表道路的长度。现在想从城市 A 到达城市 E，怎样走路程最短？最短路程的长度是多少？



【算法分析】

把 A 到 E 的全过程分成四个阶段，用 K 表示阶段变量，第 1 阶段有一个初始状态 A，有两条可供选择的支路 A-B1、A-B2；第 2 阶段有两个初始状态 B1、B2，B1 有三条可供选择的支路，B2 有两条可供选择的支路……。用 DK（X_I，$X+1_J$）表示在第 K 阶段由初始状态 X_I 到下阶段的初始状态 $X+1_J$ 的路径距离，FK（X_I）表示从第 K 阶段的 X_I 到终点 E 的最短距离，利用倒推的方法，求解 A 到 E 的最短距离。具体计算过程如下：

S1： K = 4 有

F4（D1）= 3;

F4（D2）= 4;

F4（D3）= 3;

S2： K = 3 有

F3（C1）= MIN{ D3（C1，D1）+ F4（D1），D3（C1，D2）+ F4（D2）}

= MIN{ 5+3，6+4 } = 8，

F3（C2）= D3（C2，D1）+ F4（D1）= 5+3 = 8;

F3（C3）= D3（C3，D3）+ F4（D3）= 8+3 = 11;

F3（C4）= D3（C4，D3）+ F4（D3）= 3+3 = 6;

S3： K = 2 有

F2（B1）= MIN{ D2（B1，C1）+ F3（C1），D2（B1，C2）+ F3（C2），

D2（B1，C3）+ F3(C3)} = MIN{ 1+8,6+8,3+11} = 9，

F2（B2）= MIN{ D2（B2，C2）+ F3（C2），D2（B2，C4）+ F3（C4）}

= MIN{ 8+8，4+6 } = 10;

S4： K = 1 有

F1（A）= MIN{ D1（A，B1）+ F2（B1），D1（A，B2）+ F2（B2）}

= MIN{ 5+9，3+10} = 13。

因此由 A 点到 E 点的全过程最短路径为 A→B2→C4→D3→E；最短路程长度为 13。

从以上过程可以看出，每个阶段中，都求出本阶段的各个初始状态到终点 E 的最短距离，当逆序倒推到过程起点 A 时，便得到了全过程的最短路径和最短距离。

在上例的多阶段决策问题中，各个阶段采取的决策，一般来说是与阶段有关的，决策依赖于当前状态，又随即引起状态的转移，一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义，我们称这种解决多阶段决策最优化的过程为动态规划程序设计方法。

二、动态规划的基本概念和基本模型构成

现在我们来介绍动态规划的基本概念。

1. 阶段和阶段变量：

用动态规划求解一个问题时，需要将问题的全过程恰当地分成若干个相互联系的阶段，以便按一定的次序去求解。描述阶段的变量称为阶段变量，通常用 K 表示，阶段的划分一般是根据时间和空间的自然特征来划分，同时阶段的划分要便于把问题转化成多阶段决策过程，如例题 2 中，可将其划分成 4 个阶段，即 K = 1，2，3，4。

2. 状态和状态变量：

某一阶段的出发位置称为状态，通常一个阶段包含若干状态。一般地，状态可由变量来描述，用来描述状态的变量称为状态变量。如例题 2 中，C3 是一个状态变量。

3. 决策、决策变量和决策允许集合：

在对问题的处理中作出的每种选择性的行动就是决策。即从该阶段的每一个状态出发，通过一次选择性的行动转移至下一阶段的相应状态。一个实际问题可能要有多次决策和多个决策点，在每一个阶段的每一个状态中都需要有一次决策，决策也可以用变量来描述，称这种变量为决策变量。在实际问题中，决策变量的取值往往限制在某一个范围之内，此范围称为允许决策集合。如例题 2 中，F3(C3)就是一个决策变量。

4. 策略和最优策略：

所有阶段依次排列构成问题的全过程。全过程中各阶段决策变量所组成的有序总体称为策略。在实际问题中，从决策允许集合中找出最优效果的策略成为最优策略。

5. 状态转移方程

前一阶段的终点就是后一阶段的起点，对前一阶段的状态作出某种决策，产生后一阶段的状态，这种关系描述了由 k 阶段到 k+1 阶段状态的演变规律，称为状态转移方程。

三、最优化原理与无后效性

上面已经介绍了动态规划模型的基本组成，现在需要解决的问题是：什么样的“多阶段决策问题”才可以采用动态规划的方法求解。

一般来说，能够采用动态规划方法求解的问题，必须满足最优化原理和无后效性原则：

1、动态规划的最优化原理。作为整个过程的最优策略具有：无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略的性质。也可以通俗地理解为子问题的局部最优将导致整个问题的全局最优，即问题具有最优子结构的性质，也就是说一个问题的最优解只取决于其子问题的最优解，而非最优解对问题的求解没有影响。在例题 2 最短路径问题中，A 到 E 的最优路径上的任一点到终点 E 的路径，也必然是该点到终点 E 的一条最优路径，即整体优化可以分解为若干个局部优化。

2、动态规划的无后效性原则。所谓无后效性原则，指的是这样一种性质：某阶段的状态一旦确定，则此后过程的演变不再受此前各状态及决策的影响。也就是说，“未来与过去无关”，当前的状态是此前历史的一个完整的总结，此前的历史只能通过当前的状态去影响过程未来的演变。在例题 2 最短路径问题中，问题被划分成各个阶段之后，阶段 K 中的状态只能由阶段 K+1 中的状态通过状态转移方程得来，与其它状态没有关系，特别与未发生的状态没有关系，例如从 Ci 到 E 的最短路径，只与 Ci 的位置有关，它是由 Di 中的状态通过状态转移方程得来，与 E 状态，特别是 A 到 Ci 的路径选择无关，这就是无后效性。

由此可见，对于不能划分阶段的问题，不能运用动态规划来解；对于能划分阶段，但不符合最优化原理的，也不能用动态规划来解；既能划分阶段，又符合最优化原理的，但不具备无后效性原则，还是不能用动态规划来解；误用动态规划程序设计方法求解会导致错误的结果。

四、动态规划设计方法的一般模式

动态规划所处理的问题是一个多阶段决策问题，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态；或倒过来，从结束状态开始，通过对中间阶段决策的选择，达到初始状态。这些决策形成一个决策序列，同时确定了完成整个过程的一条活动路线，通常是求最优活动路线。

动态规划的设计都有着一定的模式，一般要经历以下几个步骤：

1、划分阶段：按照问题的时间或空间特征，把问题划分为若干个阶段。在划分阶段时，注意划分后的阶段一定是有序的或者是可排序的，否则问题就无法求解。

2、确定状态和状态变量：将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然，状态的选择要满足无后效性。

3、确定决策并写出状态转移方程：因为决策和状态转移有着天然的联系，状态转移就是根据上一阶段的状态和决策来导出本阶段的状态。所以如果确定了决策，状态转移方程也就可以写出。但事实上常常是反过来做，根据相邻两段的各个状态之间的关系来确定决策。

4、寻找边界条件：给出的状态转移方程是一个递推式，需要一个递推的终止条件或边界条件。

5、程序的设计实现：一旦设计完成，实现就会非常简单。下面我们给出从初始状态开始，通过对中间阶段决策的选择，达到结束状态，按照阶段、状态和决策的层次关系，写出的程序流程的一般形式：

```
所有状态费用的初始化;
for i := 阶段最大值-1 downto 1 do     {倒推每一个阶段}
  for j := 状态最小值 to 状态最大值 do  {枚举阶段 i 的每一个状态}
    for k := 决策最小值 to 决策最大值 do {枚举阶段 i 中状态 j 可选择的每一种决策}
      begin
        f[i,j] ← min{d[i,j,k] + f[i+1,k]}
      end;
输出 f[1,1];
```

【例 9】对应的 Pascal 程序如下：

```
var  d : array[1..4,1..4,1..4] of byte;
     f : array[1..5,1..4] of byte;
     i,j,k,min : byte;
begin
  fillchar(d,sizeof(d),0);
  d[1,1,1] := 5; d[1,1,2] := 3;
  d[2,1,1] := 1; d[2,1,2] := 6; d[2,1,3] := 3;
  d[2,2,2] := 8; d[2,2,4] := 4;
  d[3,1,1] := 5; d[3,1,2] := 6;
  d[3,2,1] := 5;
  d[3,3,3] := 8;
  d[3,4,3] := 3;
  d[4,1,1] := 3;
  d[4,2,1] := 4;
  d[4,3,1] := 3;
  fillchar(f,sizeof(f),255);
  f[5,1] := 0;
  for i := 4 downto 1 do
    for j := 1 to 4 do
      for k := 1 to 4 do
        if d[i,j,k] <> 0 then
          if f[i,j] > d[i,j,k]+f[i+1,k] then f[i,j] := d[i,j,k]+f[i+1,k];
  writeln(f[1,1]);
  readln;
end.
```

【例 10】数塔问题（IOI94）有形如图 2 所示的数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一起走到底层，要求找出一条路径，使路径上的值最大。

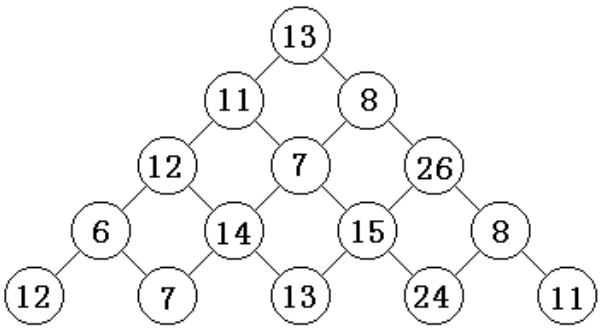


图 2

这道题如果用枚举法，在数塔层数稍大的情况下（如 40），则需要列举出的路径条数将是一个非常庞大的数目。如果用贪心法又往往得不到最优解。在用动态规划考虑数塔问题时可以自顶向下的分析，自底向上的计算。从顶点出发时到底向左走还是向右走应取决于是从左走能取到最大值还是从右走能取到最大值，只要左右两道路径上的最大值求出来了才能作出决策。同样的道理下一层的走向又要取决于再下一层上的最大值是否已经求出才能决策。这样一层一层推下去，直到倒数第二层时就非常明了。所以实际求解时，可从底层开始，层层递进，最后得到最大值。实际求解时应掌握其编程的一般规律，通常需要哪几个

关键数组来存储变化过程这一点非常重要。

一般说来，很多最优化问题都有着对应的计数问题；反过来，很多计数问题也有着对应的最优化问题。因此，我们在遇到这两类问题时，不妨多联系、多发展，举一反三，从比较中更深入地理解动态规划的思想。

其实递推和动态规划这两种方法的思想本来就很相似，也不必说是谁借用了谁的思想。关键在于我们要掌握这种思想，这样我们无论在用动态规划法解最优化问题，或是在用递推法解判定型、计数问题时，都能得心应手、游刃有余了。

【算法分析】

①贪心法往往得不到最优解：本题若采用贪心法则：13-11-12-14-13，其和为 63

但存在另一条路：13-8-26-15-24，其和为 86。

贪心法问题所在：眼光短浅。

②动态规划求解：动态规划求解问题的过程归纳为：自顶向下的分析，自底向上计算。

其基本方法是：

划分阶段：按三角形的行，划分阶段，若有 n 行，则有 n-1 个阶段，找到问题求解的最优路径。

A. 从根结点 13 出发，选取它的两个方向中的一条支路，当到倒数第二层时，每个结点其后继仅有两个结点，可以直接比较，选择最大值为前进方向，从而求得从根结点开始到底端的最大路径。

B. 自底向上计算：（给出递推式和终止条件）

①从底层开始，本身数即为最大数；

②倒数第二层的计算，取决于底层的数据：12+6=18，13+14=27，24+15=39，24+8=32；

③倒数第三层的计算，取决于底二层计算的数据：27+12=39，39+7=46，39+26=65

④倒数第四层的计算，取决于底三层计算的数据：46+11=57，65+8=73

⑤最后的路径：13——8——26——15——24

C. 数据结构及算法设计

①图形转化：直角三角形，更于搜索：向下、向右

②用三维数组表示数塔：a[x,y,1]表示行、列及结点本身数据,a[x,y,2]能够取得最大值,a[x,y,3]表示前进的方向——0 向下，1 向右；

③算法：

数组初始化，输入每个结点值及初始的最大路径、前进方向为 0；

从倒数第二层开始向上一层求最大路径，共循环 N-1 次；

从顶向下，输出路径：关键是 J 的值，由于行逐渐递增，表示向下，究竟向下还是向右取决于列的值。若 J 值比原先多 1 则向右，否则向下。

数塔问题的样例程序如下：

```
var
    a:array[1..50,1..50,1..3] of longint;
    x,y,n:integer;
begin
    write( 'please input the number of rows:');
    readln(n);
    for x:=1 to n do
        for y:=1 to x do
          begin
            read(a[x,y,1]);
            a[x,y,2]:=a[x,y,1];
            a[x,y,3]:=0
          end;
```

```
    for x:=n-1 downto 1 do
      for y:=1 to x do
          if a[x+1,y,2]>a[x+1,y+1,2] then
              begin  a[x,y,2]:=a[x,y,2]+a[x+1,y,2]; a[x,y,3]:=0 end
          else  begin a[x,y,2]:=a[x,y,2]+a[x+1,y+1,2];a[x,y,3]:=1 end;
    writeln('max=',a[1,1,2]);
    y:=1;
    for x:=1 to n-1 do
      begin
        write(a[x,y,1],'-> ');
        y:=y+a[x,y,3]
      end;
    writeln(a[n,y,1])
end.
```

【输入】:

```
5    {数塔层数}
13
11   8
12   7    26
6    14   15    8
12   7    13   24    11
```

【输出】max=86

13—8—26—15—24

【例 11】求最长不下降序列

㈠问题描述：设有由 n 个不相同的整数组成的数列，记为:b(1)、b(2)、……、b(n)且 b(i)<>b(j) (i<>j)，若存在 i1<i2<i3< … < ie 且有 b(i1)<b(i2)< … <b(ie)则称为长度为 e 的不下降序列。程序要求，当原数列给出之后，求出最长的不下降序列。

例如 13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中 13，16，18，19，21，22，63 就是一个长度为 7 的不下降序列，同时也有 7 ，9，16，18，19，21，22，63 长度为 8 的不下降序列。

㈡算法分析：根据动态规划的原理，由后往前进行搜索。

1．对 b(n)来说，由于它是最后一个数，所以当从 b(n)开始查找时，只存在长度为 1 的不下降序列；

2．若从 b(n-1)开始查找，则存在下面的两种可能性：

①若 b(n-1)<b(n)则存在长度为 2 的不下降序列 b(n-1)，b(n)。

②若 b(n-1)>b(n)则存在长度为 1 的不下降序列 b(n-1)或 b(n)。

3．一般若从 b(i)开始，此时最长不下降序列应该按下列方法求出:

在 b(i+1),b(i+2),…,b(n)中，找出一个比 b(i)大的且最长的不下降序列，作为它的后继。

㈢数据结构：为算法上的需要，定义一个数组整数类型二维数组 b(N,3)

1．b(I,1)表示第 I 个数的数值本身；

2．b(I,2)表示从 I 位置到达 N 的最长不下降序列长度

3．b(I,3)表示从 I 位置开始最长不下降序列的下一个位置，若 b[I,3]=0 则表示后面没有连接项。

㈣求解过程：

①从倒数第二项开始计算，后面仅有 1 项，比较一次，因 63>15，不符合要求，长度仍为 1。

②从倒数第三项开始其后有 2 项，需做两次比较，得到目前最长的不下降序列为 2，如下表：

	11	12	13	14	……		11	12	13	14
		22	63	15	……		21	22	63	15
		2	1	1	……		3	2	1	1
		13	0	0	……		12	13	0	0

㈤一般处理过程是：

①在 i+1,i+2,…,n 项中，找出比 b[I,1]大的最长长度 L 以及位置 K;

②若 L>0，则 b[I,2]:=L+1;b[I,3]:=k;

最后本题经过计算，其数据存储表如下：

1	2	3	4	5	6	7	8	9	10	11	12	13	14
13	7	9	16	38	24	37	18	44	19	21	22	63	15
7	8	7	6	3	4	3	5	2	4	3	2	1	1
4	3	4	8	9	7	9	10	13	11	12	13	0	0

初始化:

```
for i:=1 to n do
    begin
      read(b[i,1]);
      b[i,2]:=1;b[i,3]:=0;
     end;
```

下面给出求最长不下降序列的算法:

```
for i:=n-1 downto 1 do
      begin
         L:=0;k:=0;
         for j:=i+1 to n do
         if(b[j,1]>b[i,1])and(b[j,2]>L) then begin
                                               L:=b[j,2];k:=j;
                                             end;
         if L>0 then begin
                        b[i,2]:=L+1;b[i,3]:=k;
                     end;
       end;
```

下面找出最长不下降序列：

```
L:=1;
for j:=2 to n do
  if b[j,2]>b[L,2] then L:=j;
```

最长不下降序列长度为 B(L, 2)序列

```
while L<>0  do
  begin
    write(b[L,1]:4);
    L:=b[L,3];
  end;
```

【参考程序】

```
var
    n,i,L,k,j:integer;
    b:array[1..100,1..3]of integer;
begin
  writeln('input n:');
  readln(n);
  for i:=1 to n do
    begin
      read(b[i,1]);
      b[i,2]:=1;b[i,3]:=0;
    end;
  for i:=n-1 downto 1 do
      begin
         L:=0;k:=0;
         for j:=i+1 to n do
         if(b[j,1]>b[i,1])and(b[j,2]>L) then begin
                                               L:=b[j,2];
                                               k:=j;
                                             end;
         if L>0 then begin
                       b[i,2]:=L+1;b[i,3]:=k;
                     end;
      end;
      L:=1;
      for j:=2 to n do
         if b[j,2]>b[L,2] then L:=j;
      writeln('max=',b[L,2]);
        while L<>0  do
          begin
            write(b[L,1]:4);
            L:=b[L,3];
          end;
        writeln;
        readln;
end.
```

程序运行结果：

【输入】:

13 7 9 16 38 24 37 18 44 19 21 22 63 15

【输出】:

```
max=8
7   9   16   18   19   21   22   63
```